Universidade do Minho
Escola de Engenharia

Rui Pedro Delgado Sousa

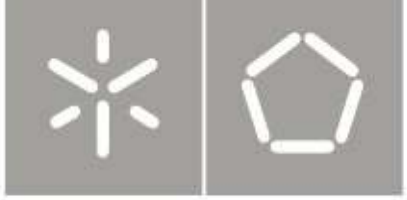
Micro-Autómato programável com ambiente de desenvolvimento multilinguagem

Julho de 2014

Micro-Autómato programável com ambiente de desenvolvimento multilinguagem

Rui Pedro Delgado Sousa

UMinho | 2014



Universidade do Minho
Escola de Engenharia

Rui Pedro Delgado Sousa

Micro-Autómato programável com ambiente de desenvolvimento multilinguagem

Dissertação de Mestrado
Ciclo de Estudos Integrados Conducentes ao
Grau de Mestre em Engenharia Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do
Professor Doutor Agostinho Gil Teixeira Lopes

Julho de 2014

DECLARAÇÃO

Autor: Rui Pedro Delgado Sousa

Correio eletrónico: a58772@alunos.uminho.pt

Telemóvel: 969179725

Número do cartão de cidadão: 13837229 2ZZ5

Título da dissertação: Micro-Autómato programável com ambiente de desenvolvimento multilinguagem

Ano de conclusão: 2014

Orientador: Professor Doutor Agostinho Gil Teixeira Lopes

Designação do Mestrado: Mestrado em Engenharia Eletrónica Industrial e Computadores

Ciclo de Estudo Integrados Conducentes ao Grau de Mestre em Engenharia

Área de Especialização: Robótica

Escola: Universidade do Minho – Azurém

Departamento: Departamento de Eletrónica Industrial

Guimarães, _____ / _____ / _____

Assinatura: _____

Agradecimentos

O presente documento marca a conclusão deste capítulo tão importante a nível académico e pessoal, que envolveu o apoio de algumas pessoas a quem pretendo deixar um agradecimento.

Em primeiro lugar, gostaria de agradecer ao meu orientador, Agostinho Gil Teixeira Lopes pela orientação e enriquecedoras trocas de ideias com que me presenteou ao longo desta dissertação.

Em segundo lugar, gostaria de agradecer aos meus colegas do Laboratório de Automação e Robótica da Universidade do Minho, que acompanharam o meu trabalho incentivando a sua realização, bem como aos funcionários das oficinas do Departamento de Eletrónica Industrial pelo apoio prestado a nível da construção do *hardware*.

Gostaria também de agradecer ao Departamento de Eletrónica Industrial, nomeadamente ao Laboratório de Automação e Robótica da Universidade do Minho pelo financiamento e disponibilidade manifestada na cedência de meios materiais, que se revelaram necessários à construção dos protótipos.

Por último e não menos importante, quero agradecer aos meus familiares em especial aos meus pais, Sónia Delgado e Manuel Sousa por todo o apoio prestado ao longo destes anos como estudante, há minha namorada e meus amigos pela forma como souberam compreender as minhas ausências e horários pouco ortodoxos.

Resumo

Representando atualmente o cérebro em ambientes industriais, os Autómatos Programáveis já se encontram fortemente implementados em inúmeros tipos de indústrias. A necessidade da existência destes, em ambientes industriais deve-se ao facto de, anteriormente, os complexos sistemas de controlo serem baseados em relés mecânicos de elevado custo que, ao contemplarem uma vida limitada, obrigavam à sua manutenção periódica. Assim, o principal problema tinha origem na eventual necessidade de alteração dos requisitos de produção, o que obrigava à alteração do sistema a controlar, originando consequentemente maiores custos de produção.

Atualmente os Autónomos Programáveis consistem num equipamento eletrónico programável por técnicos dotados de conhecimentos em programação, com a finalidade de controlar em tempo real, máquinas ou processos sequenciais.

Neste sentido, o principal objetivo deste trabalho é o desenvolvimento de um Autómato Programável de dimensões reduzidas - um Micro-Autómato Programável. Assentando na criação de um *hardware* de dimensões reduzidas, este Micro-Autómato Programável dispõe de um número de portas de entrada/saída também reduzido - 4 entradas e 4 saídas, e utiliza um microprocessador de pequenas dimensões. O *Software* de Programação e Ambiente de Desenvolvimento Multilinguagem deste Micro-Autómato Programável foi desenvolvido com recurso à linguagem C# o que permite ao técnico o desenvolvimento da sua aplicação para o Micro-Autómato em três linguagens: *C, Ladder Diagram* e *Sequential Function Chart* – GRAFCET, podendo optar-se por aquela na qual o técnico se sente mais à vontade ao nível dos seus conhecimentos.

Esta dissertação obteve como resultado, um Micro-Autómato Programável com ambiente de desenvolvimento multilinguagem, tendo sido criado um protótipo de fácil integração por parte do técnico responsável, de baixo custo e que desempenhe todas as funções necessárias a um Autómato Programável dispondo de um *Software* de Programação próprio.

Palavras-chave: *PLC*, Autómato Programável, Multilinguagem, *Ladder*, *Grafcet*

Abstract

Programmable automation technology which currently represents the brain in industrial environments is already heavily deployed in many types of industries. The need for this kind of technology in industrial environments is due to the fact that previously complex control systems were based on high cost mechanical relays that required regular maintenance and had a limited life span. Thus, the main problem arose from the possible need to change the production requirements which required modifications in the control system, consequently resulting in higher production costs.

Programmable automation currently consists of electronic equipment programmable by technicians equipped with knowledge in programming, in order to control machines or sequential processes in real time.

In this sense, the main objective of this study was to develop a programmable automation of reduced dimensions - a micro-programmable automation. The study aimed to create hardware with reduced dimensions in terms of the number of inputs and outputs - 4 of each - and in terms of the small size of its microprocessor. Software and Programming Environment Multilanguage Development of this programmable micro-automation was developed using the C # language which allows the technician to develop his or her applications for micro-automation in three languages (C, Ladder Diagram and Function Sequential Chart – GRAFCET) thus enabling the technician to choose the language he or she is most comfortable with.

This study resulted in the production of a micro-programmable automation with a multi-language development environment and a prototype that can be easily integrated by the technician responsible. The automation can be acquired at low cost and will perform all the necessary functions of a programmable automation whilst also providing its own software programme.

Keywords: PLC, Programmable Automation Controller, Multilanguage, Ladder, Grafcet

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas, Siglas e Nomenclaturas

IEC - *International Electrotechnical Commission*

USB – *Universal Serial Bus*

ABB – *Asea Brown Boveri*

CPU – *Central Processing Unit*

RAM – *Random-Access Memory*

EPROM – *Erasable Programmable Read Only Memory*

EEPROM – *Electrically-Erasable Programmable Read Only Memory*

ADC – *Analog-to-Digital Converter*

Hz – *Hertz*

A – Ampére

V – Volt

RX – *Receiver*

TX – *Transmitter*

GND – *Ground*

PWM – *Pulse Width Modulation*

LED – *Light Emitting Diode*

AC – *Alternating Current*

DC – *Direct Current*

ICSP – *In Circuit Serial Programming*

UML – *Unified Modeling Language*

Capítulo 1

1. Introdução

Esta dissertação surge no âmbito do Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores da Universidade do Minho e tem por objetivo documentar o trabalho desenvolvido durante o projeto de dissertação denominado Micro-Autómato Programável com Ambiente de Desenvolvimento Multilinguagem.

Esta introdução pretende expor o problema que está na origem deste trabalho, aludindo e enquadrando a motivação pessoal na realização do mesmo.

Nuna fase posterior serão enunciados os objetivos que se propõem alcançar no final desta dissertação, devidamente calendarizados.

1.1 Problema de Investigação

Atualmente os Autómatos Programáveis consistem num equipamento eletrónico programável por técnicos dotados de conhecimentos em programação, com a finalidade de controlar, em tempo real, máquinas ou processos. Assim, as características funcionais assentam na adaptação ao ambiente industrial, numa programação própria orientada para a automação e num funcionamento síncrono com a execução cíclica do programa.

Observando o mercado disponível de Autómatos Programáveis verifica-se que existem mais de 10 marcas com vários modelos disponíveis, sendo o modelo mais barato um Autómato Programável pertencente à Siemens, modelo 6ES72741XH000X4, cujo preço ronda em 2014, os 120 € [1]. Este Autómato dispõe de 14 portas de entrada/saída - 6 de entrada e 8 de saída, tendo como dimensões 9 cm de comprimento, 4 cm de altura e 2,5 cm de profundidade e podendo ser programado exclusivamente em *Ladder Diagram* com interface série que se encontra em fase de

extinção [2]. Quanto ao modelo mais caro, trata-se de um Autómato Programável pertencente a Omron, modelo CP1H CPU 40, cujo preço ronda em 2014, os 2740€ [3]. Este Autómato dispõe de 40 portas de entrada/saída - 24 de entrada e 16 de saída, tendo como dimensões 15 cm de comprimento, 11 cm de altura e 8,5 cm de profundidade e podendo ser programado exclusivamente em *Ladder Diagram* com interface série, Usb e comunicação de rede por Ethernet [4].

Ao adquirir um Autómato Programável é sempre necessário, para proceder a sua programação, a compra de um *Software* de Programação da referida marca com compatibilidade com o respetivo Autómato. Assim, o *Software* de Programação para o Autómato Programável da Siemens, acima referido, tem um custo atualmente aproximado de 336€ para uma única licença vitalícia [5]. Por outro lado, o *Software* de Programação para o Autómato Programável da Omron tem um custo atualmente aproximado de 1419€ para a versão total [6].

1.2 Enquadramento e Motivação

A estrutura de um Autómato Programável apresenta-se de forma compacta onde todos os elementos se reúnem num só bloco, ou de forma modular onde cada elemento representa um bloco individual com um determinado tipo de função. Assim um Autómato Programável engloba:

- Memória - onde é guardado o programa, os estados internos e salvaguardados os dados e as variáveis internas;
- Unidade Central de Processamento (*CPU*);
- Fonte de Alimentação;
- Interfaces de Entrada e Saída;
- Interfaces de Comunicação.

Com a crescente complexidade na programação dos Autómatos Programáveis existiu a necessidade da normalização de Linguagens de Programação como norma adequada às várias aplicações existentes, tendo ficado definidas cinco linguagens de programação - gráficas e textuais [7].

- *Instruction List;*
- *Structured Text;*
- *Function Block Diagram;*
- *Ladder Diagram;*

- *Sequential Function Chart;*

Perante as soluções encontradas no mercado, o que se propõe com este trabalho é, como já foi referido anteriormente, o desenvolvimento um Autómato Programável de dimensões reduzidas - um Micro-Autómato Programável. Como resultado espera-se, então, a obtenção de um Micro-Autómato Programável, de ambiente de desenvolvimento multilinguagem - *Software* de Programação, sendo o objetivo final a criação um protótipo de fácil integração por parte do técnico responsável.

1.3 Objetivos

Os objetivos deste projeto passam pela criação de um *hardware* de dimensões reduzidas, dispondo de um número de portas de entrada/saída também reduzido - 4 entradas e 4 saídas e utilizando um microprocessador de dimensões também reduzidas. A interface de comunicação será em micro Usb, uma interface muito utilizada atualmente por ser extremamente prática.

O *Software* de Programação deste Micro-Autómato Programável será desenvolvido com recurso à linguagem C#. Este *Software* de Programação irá permitir ao técnico o desenvolvimento da sua aplicação para o Micro-Autómato em três linguagens: *C, Ladder Diagram* e *Sequential Function Chart – GRAFCET*, podendo escolher aquela na qual se sente mais à vontade ao nível dos seus conhecimentos.

Após a finalização do projeto e a assunção da inexistência de erros por parte do técnico e depurados pelo *software*, as linguagens serão passadas para uma linguagem de mais baixo nível - código C, com o objetivo de ser compilado e ser criado um ficheiro binário para posterior programação do Micro-Autómato. O *Software,* através de uma interface amigável com o utilizador, permitirá uma fácil implementação do projeto e uma simples programação do Micro-Autómato.

1.4 Estrutura do Documento

O documento encontra-se dividido em seis capítulos. O capítulo 1 consiste numa introdução onde é apresentado o projeto, explicado e enquadrado o problema de investigação e apresentadas as motivações que estiveram na origem deste trabalho. É igualmente neste capítulo que se expõe os objetivos a alcançar para o sucesso do projeto.

No capítulo 2, intitulado "Estado de Arte", é feita a análise sobre o estado da arte analisando as propostas de mercado disponíveis até ao momento, no âmbito deste trabalho. Ainda neste capítulo é apresentada a norma IEC 61131-3 que define as linguagens utilizadas para a programação de Autómatos Programáveis.

No capítulo 3, "Fundamentos Teóricos", são apresentados os conceitos teóricos necessários para a correta compreensão dos temas ao longo deste trabalho, deste a estrutura de um autómato programável até as linguagens utilizadas no projeto.

O capítulo 4, intitulado "Estrutura do Sistema", onde é apresentada a estrutura desenvolvida para a implementação da solução de *hardware* e *software* tem o propósito de atingir os objetivos propostos para o derradeiro sucesso deste projeto.

No capítulo 5, "Resultados Obtidos e Discussão", como o próprio nome indica, são apresentados detalhadamente os resultados obtidos nos vários testes realizados bem como a discussão dos mesmos.

Finalmente, no capítulo 6, "Conclusão e Trabalho Futuro", são expostas as conclusões finais desta dissertação sugerindo algumas propostas para trabalhos futuros.

Nos apêndices foram colocados os códigos em linguagem C gerados pelo ambiente de desenvolvimento para os vários testes elaborados no capítulo 5.

Nos anexos foi ainda colocado o livro de exercícios utilizado para realização dos testes a nível de *hardware* e *software* apresentados também no capítulo 5.

Capítulo 2

2. Estado de Arte

Neste capítulo serão analisadas propostas de mercado, disponíveis até ao momento, no âmbito deste trabalho, tendo por objetivo fundamentar teoricamente as várias soluções de Autómatos Programáveis.

Serão, então, apresentadas as principais marcas disponíveis no mercado, e os seus produtos, tanto a nível de *hardware* como *software*. Será igualmente apresentada a norma IEC 61131-3 que define as linguagens utilizadas para a programação de Autómatos Programáveis.

2.1 IEC 61131-3

A IEC 61131-3 consiste na terceira parte da norma internacional IEC 61131 para Autómatos Programáveis, tendo sido publicada pela primeira vez em Dezembro de 1993 pela IEC, contando já com uma terceira edição, publicada em Fevereiro de 2013 [8].

A norma IEC 61131-3 estabelece as linguagens de programação definindo, assim, duas linguagens gráficas, duas linguagens de padrão textual e uma linguagem de organização de elementos para programação sequencial. Assim, estas cinco linguagens são [7]:

2.1.1. *Instruction List*

A Lista de Instruções, *Instruction List*, é uma linguagem de muito baixo nível cuja sequência de instruções é criada e executada linha a linha, estrutura semelhante à linguagem Assembler usada em microprocessadores.

A vantagem desta linguagem consiste na construção de aplicações complexas através da disposição de um conjunto de funções básicas, sendo possível criar aplicações com código mais otimizado do que em outras linguagens.

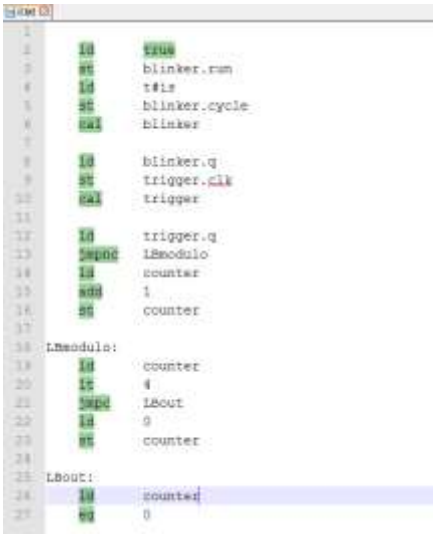


Figura 1 - *Instruction List*

Por outro lado, esta linguagem apresenta como desvantagem um maior tempo de demora na elaboração dos programas inclusive em pequenas aplicações.

2.1.2. *Structured Text*

O Texto Estruturado, *Structured Text*, é uma linguagem de baixo nível, mais simples que a anterior, o que a torna numa linguagem fácil de utilizar e interpretar. A sua estrutura é semelhante à da linguagem Pascal, também muito utilizada em programação.

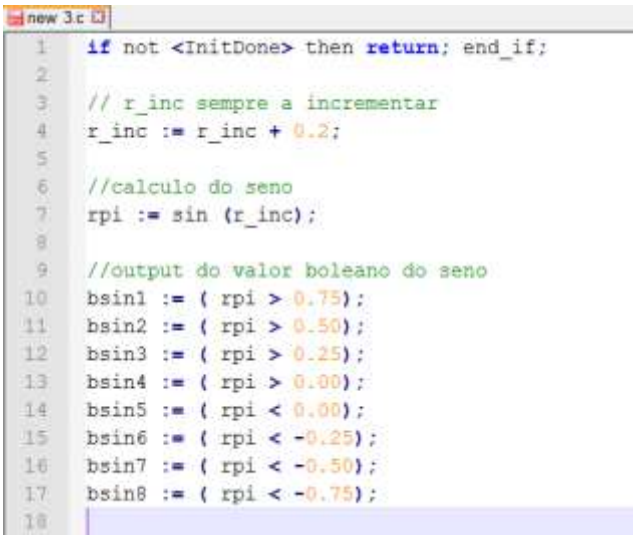


Figura 2 - *Structured Text*

A vantagem desta linguagem consiste na facilidade de criar aplicações que operam com variáveis de diferentes tipos de dados, valores analógicos, digitais, entre outros, o que a torna na linguagem mais adequada para a implementação de algoritmos matemáticos complexos.

2.1.3. *Function Block Diagram*

O Diagrama de Funções, *Function Block Diagram*, é uma linguagem que permite o desenvolvimento de programas num ambiente gráfico dispondo da utilização de blocos de funções existentes na biblioteca.

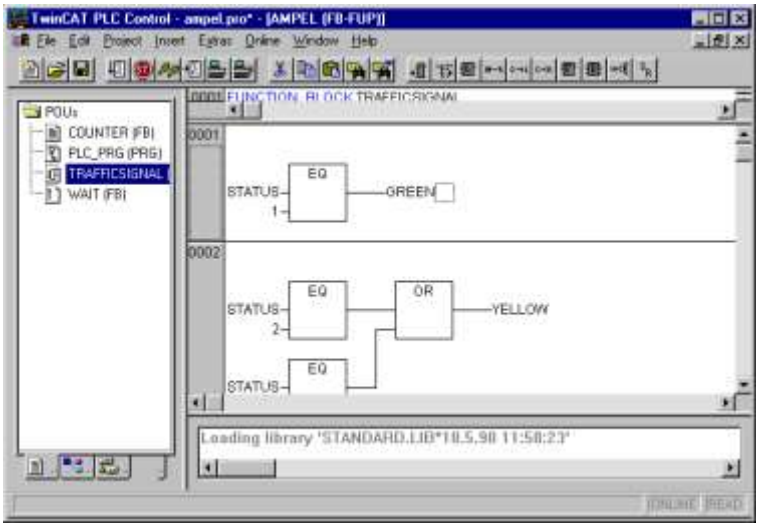


Figura 3 - *Function Block Diagram* [9]

Apresenta como vantagem a utilização de uma simbologia equivalente aos circuitos de portas lógicas, o que resulta numa linguagem cómoda de utilizar por entendidos no ramo da eletrónica.

2.1.4. *Ladder Diagram*

O Diagrama de Contactos, *Ladder Diagram*, é uma linguagem de programação gráfica que permite a inserção de símbolos como relés, contactos e blocos de funções, sendo por isso uma linguagem versátil.

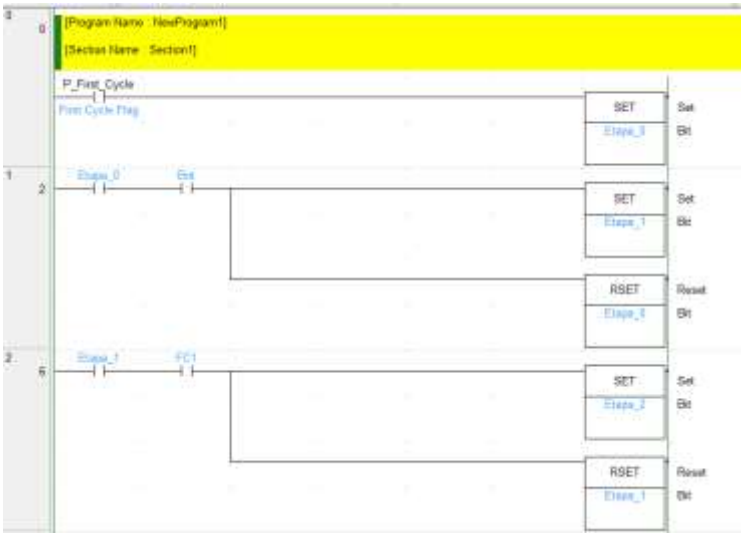


Figura 4 - *Ladder Diagram*

A vantagem desta linguagem relaciona-se com a abrangência de fabricantes que a utiliza para a programação nos seus autómatos, tornando-a na linguagem mais utilizada pelos técnicos de programação industrial.

2.1.5. *Sequential Function Chart*

O Gráfico Sequencial de Funções, *Sequential Function Chart* ou *Grafcet*, é uma linguagem gráfica que apresenta o processo na forma de um diagrama constituído por um conjunto de etapas e transições. Assim, caso sejam cumpridas as condições nas transições, o automatismo fica sequencial ativando as ações nas respetivas etapas.

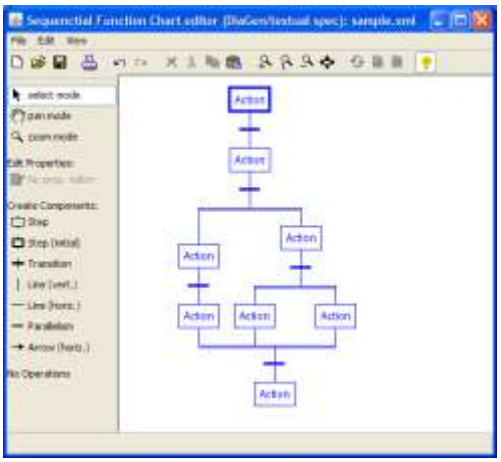


Figura 5 - *Sequential Function Chart* [10]

A vantagem desta linguagem deve-se à utilização de símbolos gráficos e consiste na criação de diagramas que representam algoritmos com boa legibilidade na análise funcional.

2.2 ABB

A ABB é uma empresa multinacional Suíça, criada em 1988, considerada líder em tecnologia de energia e automação. O Grupo ABB opera em mais de 100 países contando com mais de 130 mil trabalhadores [11].

Figura 6 - Logotipo da ABB [13]

A Divisão de Automação da ABB oferece produtos e serviços para a produção industrial incluindo motores elétricos, geradores, autómatos programáveis, robôs industriais, geradores de energia eólica e inversores de energia solar [12].

O foco principal da ABB é fornecer aos clientes, sistemas de controlo automático, otimização do plano de produção e aplicações de automação específicos para a indústria.

2.2.1. Autómatos Programáveis

A ABB dispõe de vários Autómatos Programáveis no mercado pelo que serão apenas analisadas duas soluções desta empresa. O modelo AC500 PM554-T pelo facto de ser o produto economicamente mais barato, e o modelo AC500 PM564-T por representar uma solução mais completa embora se apresente como um dos modelos mais dispendiosos desta empresa [14].

Figura 7 - AC500 PM554-T [15]

Figura 8 - AC500 PM564-R [16]

A tabela 1 expõe os dois modelos selecionados permitindo a comparação das características de ambos. Observa-se, então, a existência de uma elevada discrepância no que toca ao valor de mercado dos dois modelos, devido ao facto da elevada diferença do número e tipo de portas de entrada/saída. Enquanto o modelo economicamente mais acessível se encontra limitado apenas a portas digitais e em pequeno número, o modelo mais dispendioso tem um elevado número de portas, tanto digitais como analógicas. Outra diferença observada entre estes dois modelos é que o primeiro, o mais económico, aceita a programação em 5 linguagens enquanto o segundo aceita apenas uma linguagem. Logo, o primeiro é mais abrangente no que toca à escolha da programação por parte do técnico. Pode também dizer-se que o modelo mais dispendioso tem maior número de portas de entrada/saída, apresentando dimensões mais elevadas em comparação com o outro.

Tabela 1 - Modelos selecionados da ABB

	AC500 PM554-T	AC500 PM564-R
Dimensões (cm)		
Largura	8,2	8,2
Comprimento	13,5	14,8
Profundidade	7,4	7,4
Tensões		
Alimentação	24 V DC	100-240 V AC
Entrada Digital	24 V DC	24 V DC
Saída Digital	24 V DC	100-240 V AC
Entrada Analógica	-	0-10 V (10bits)
Saída Analógica	-	0-10 V
Corrente (A)		
Saída Digital	0,5 A	2A
Saída Analógica	-	0-20 mA
Portas		
Entradas Digitais	8	328
Saídas Digitais	6	246
Entradas Analógicas	-	162
Saídas Analógicas	-	161
Tipo de Saída	Transístor	Analógica/Relé
Entradas/Saídas Máximas	630 (406 Digitais,224 Analógicas)	880 (560 Digitais,320 Analógicas)
Memória (kB)		
Programa	128	128
Dados	14	14
Temperatura (°C)		
Máxima de Funcionamento	60	60
Mínima de Funcionamento	0	0
Programação		
Tipos de Linguagem	*Instruction List, Structured Text, Function Block Diagram, Ladder e Grafcet*	*ANSI-C, C*
Porta de Programação	RS485	RS485
Software Próprio	Não	Não
Preço (€)		
Valor no RS	202,84	707,14

2.2.2. *Software* de Programação

A ABB não dispõe de Software de Programação próprio para os seus Autómatos Programáveis optando, assim, por usar o *IEC 61131-3 Programming Tools,* uma solução da empresa NovaTech.

Esta solução permite a criação e o desenvolvimento do programa de controlo do Autómato Programável nas 5 linguagens de programação normalizadas [17].

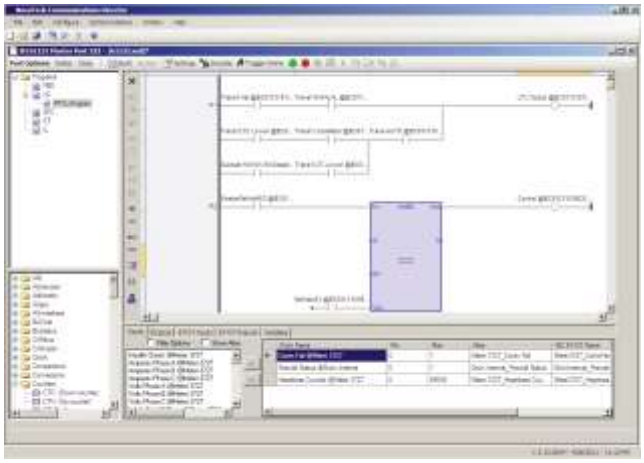


Figura 9 - IEC 61131-3 Programming Tools [18]

O *IEC 61131-3 Programming Tools* engloba as 5 linguagens de programação: *Instruction List*, *Structured Text*, *Function Block Diagram*, *Ladder* e *Grafcet,* assim como as suas específicas sintaxes e semânticas.

2.3 Mitsubishi

A Mitsubishi, empresa multinacional Japonesa criada em 1870, após um período de diversificação, resultou na criação de três entidades: a Mitsubishi *Bank*, fundada em 1919, tornando-se no maior banco do Japão resultado de duas fusões, em 1996 com o Banco de Tóquio e em 2004 com a *UFJ Holding*; a Mitsubishi *Corporation*, fundada em 1950, sendo atualmente a maior empresa comercial do Japão e a Mitsubishi *Heavy Industries* que inclui três empresas industriais: a Mitsubishi *Motors* - o 6° maior construtor automóvel japonês; a Mitsubishi *Atomic Industry* - uma empresa de potência nuclear e a Mitsubishi *Chemical* – a maior empresa química japonesa [19].

MITSUBISHI ELECTRIC
FACTORY AUTOMATION

Figura 10 - Logotipo da Mitsubishi *Electric* [21]

Com a divisão da Mitsubishi *Electric* surge a oferta de produtos em diversas áreas como sistemas para construção (ar condicionado e elevadores) e sistemas de comunicação (sistemas de segurança e sistemas espaciais) [20].

Na área automação industrial a Mitsubishi Electric disponibiliza da oferta em sistemas de automação e maquinaria de automação industrial.

2.3.1. Autómatos Programáveis

Dado que a Mitsubishi dispõem de vários Autómatos Programáveis no mercado, serão apenas analisadas duas soluções desta empresa: o modelo FX1S-10MT-DS [22], um produto economicamente mais barato e o modelo FX1N24MR-ES/UL [23], um dos modelos mais dispendiosos desta empresa.

Figura 11 - FX1S-10MR-DS [24]

Figura 12 - FX1N24MR-ES/UL [25]

Na tabela 2, estão expostos os dois modelos referidos e comparadas as suas características. Observa-se, então, que existe uma elevada discrepância no que toca ao valor dos dois modelos, o que se justifica pela diferença do número de portas de entrada/saída e da memória de programa disponível. Enquanto o modelo economicamente mais acessível está limitado apenas a 10 portas de entrada/saída, o modelo mais dispendioso tem 24 portas de entrada/saída, podendo ainda ser expandido até 32. Outra diferença entre este dos modelos é que o modelo mais acessível está limitado a uma memória de programa de 2000 passos, enquanto o modelo mais dispendioso está limitado a uma memória de programa de 8000 passos, o que permite programas mais complexos.

Devido ao facto do modelo mais dispendioso ter um maior número de portas de entrada/saída, este modelo tem também dimensões mais elevadas quando comparado com o outro modelo.

Tabela 2 - Modelos selecionados da Mitsubishi Electric

	FX1S-10MR-DS	FX1N24MR-ES/UL
Dimensões (cm)		
Largura	6	9
Comprimento	9	9
Profundidade	4,9	7,5
Tensões		
Alimentação	24 V DC	100-240 V AC
Entrada Digital	24 V DC	24 V DC
Saída Digital	250 V AC / 30 V DC	240 V AC / 30 V DC
Corrente (A)		
Saída Digital	2A	2A
Portas		
Entradas Digitais	6	14
Saídas Digitais	4	10
Tipo de Saída	Relé	Relé
Entradas/Saídas Máximas	10	32
Memória		
Programa	2000 Passos	8000 Passos
Dados	256 Registos	256 Registos
Temperatura (°C)		
Máxima de Funcionamento	55	55
Mínima de Funcionamento	0	0
Programação		
Tipos de Linguagem	*Ladder Logic*	*Ladder Logic*
Porta de Programação	RS422	RS422
Software Próprio	Sim	Sim
Preço (€)		
Valor no RS	123,46	443,77

2.3.2. *Software* de Programação

A Mitsubishi dispõe de um *Software* de Programação próprio, chamado PLC *Software* GX Works2 FX. Com o objetivo de máxima eficiência, este *Software* permite aos técnicos de programação combinar entre as cinco diferentes linguagens de programação, possibilitando o armazenamento de partes dos projetos desenvolvidos e criando uma biblioteca para uso futuro [26].

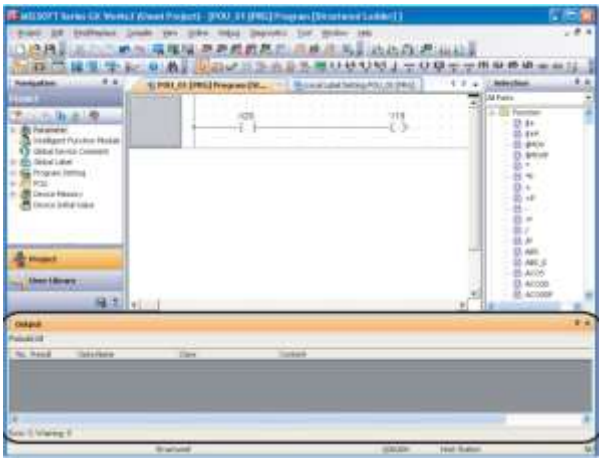


Figura 13 - PLC Software GX Works2 FX [27]

A função PLC virtual permite fazer a simulação completa do sistema antes da montagem do *hardware* no local. Este *Software* de Programação, apesar de ter um custo consideravelmente elevado, é uma ferramenta essencial no desenvolvimento de projetos, sendo versátil e muito completa.

2.4 Omron

A Omron, empresa multinacional Japonesa de componentes eletrônicos, foi fundada em 1933. O seu ramo de negócios primários assenta na fabricação e venda de componentes de automação industrial e equipamentos médicos como: termômetros, monitores de pressão arterial e nebulizadores [28].

Figura 14 - Logotipo da Omron [29]

Na área da automação industrial, a Omron oferece produtos como: sensores, autómatos programáveis, interfaces homem máquina, inversores, servos motores, controladores de temperatura, temporizadores, contadores, cortinas de segurança, *scanners* de área, sensores de medição a laser e sistemas de visão industrial.

2.4.1. Autómatos Programáveis

Tendo a Omron ao dispor, no mercado, vários Autómatos Programáveis, para prossecução deste projeto serão apenas escolhidas duas soluções desta empresa: o modelo CP1E-E [30], pelo facto de ser o produto mais limitado e por isso mais económico e o modelo CP1H CPU 40 [4], por representar a solução mais completa desta empresa, embora mais dispendiosa.

Figura 15 - CP1E-E [31]

Figura 16 - CP1H CPU 40 [32]

Na tabela 3, são expostos os dois modelos selecionados e comparadas as suas características. Observa-se, então, uma elevada diferença no que diz respeito ao custo dos dois modelos, justificada pelo facto da solução economicamente mais vantajosa apenas ter disponíveis 20 portas de entrada/saída, todas digitais e a memória disponível de programa ser apenas de 2000 passos. Por outro lado, o modelo mais dispendioso dispõe de 40 portas de entrada/saída analógicas e digitais, podendo ser expandidas até 320 portas e a memória disponível de programa é 10 vezes superior ao outro modelo - 20000 passos. Pelo facto do número possível de portas de entrada/saída ser substancialmente mais elevado, no modelo mais dispendioso, a sua memória de dados/registos é superior bem como as suas dimensões são mais elevadas quando comparando com o outro modelo.

Tabela 3 - Modelos selecionados da Omron

	CP1E-E 100-240Vac	CP1H CPU 40
Dimensões (cm)		
Largura	11	11
Comprimento	8,6	15
Profundidade	8,5	8,5
Tensões		
Alimentação	100-240 V AC	100-240V AC
Entrada Digital	24 V DC	24 V DC
Saída Digital	250 V AC / 30 V DC	100-240V AC
Entrada Analógica	-	0-10 V (1/4000 resolução)
Saída Analógica	-	0-10 V
Corrente (A)		
Saída Digital	2A	2A
Saída Analógica	-	4-20 mA
Portas		
Entradas Digitais	12	24
Saídas Digitais	8	16
Entradas Analógicas	-	4
Saídas Analógicas	-	2
Tipo de Saída	Relé	Analógica/Relé
Entradas/Saídas Máximas	20	320
Memória (kB)		
Programa	8 KB/2 000 passos	20KB/20 000 passos
Dados	2 000 Palavras	32KB
Temperatura (°C)		
Máxima de Funcionamento	55	55
Mínima de Funcionamento	0	0
Programação		
Tipos de Linguagem	*Ladder Logic*	*Ladder Logic*
Porta de Programação	Usb 1.1	Usb 1.1/Ethernet
Software Próprio	Sim	Sim
Preço (€)		
Valor no RS	199,75	2743,2

2.4.2. *Software* de Programação

A Omron dispõe de um *Software* de Programação próprio, chamado CX-*One*. O *Software* de Programação da Omron permite aos técnicos industriais configurar e programar um conjunto de Autómatos Programáveis, sistemas de interface homem/máquina e sistemas de controlo de redes industriais. Com este *software* é possível reduzir substancialmente a complexidade da

configuração permitindo criar sistemas de automação industrial sem necessitar de grandes conhecimentos.

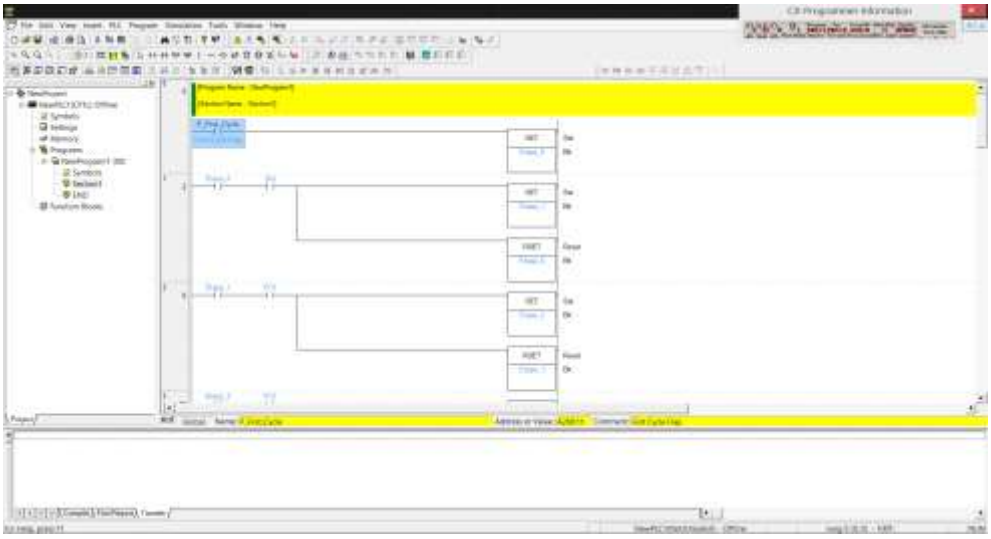


Figura 17 - Cx-One

A criação de programas é feita com recurso a linguagem *Ladder*, sendo esta uma linguagem de programação gráfica que permite a inserção de símbolos com relés, contactos e blocos de funções o que a torna numa linguagem versátil [33].

2.5 Siemens

A Siemens é uma empresa multinacional Alemã, fundada em 1847, que possui três principais áreas de negócio: Indústria; Energia e Medicina. Iniciou a sua atividade no fabrico de equipamentos de telecomunicações e atualmente está nas áreas de material elétrico, setor energético (elétrico e nuclear), equipamento hospitalar, painéis solares e computadores.

Figura 18 - Logotipo da Siemens [34]

A Siemens oferece produtos no setor industrial para cinco divisões: automação industrial; tecnologia automóvel; tecnologia na construção; soluções industriais e mobilidade. Também no setor da energia, oferece produtos em cinco divisões: geração de energia fóssil; energia renovável; óleo e gás; transmissão de energia e distribuição de energia [35].

2.5.1. Autómatos Programáveis

A Siemens dispõe de vários Autómatos Programáveis no mercado, desta vasta gama serão apenas analisadas duas soluções desta empresa. O modelo CPU1211C [36] pelo facto de a ser o produto mais acessível economicamente e o modelo CPU314C-2PN/DP [37] sendo a solução mais dispendiosa desta empresa.

Figura 19 - Siemens CPU1211C [38]

Figura 20 - Siemens CPU314C-2PN/DP [39]

Na tabela 4 estão expostos os dois modelos, selecionadas e comparadas as suas características. Observa-se que existe uma elevada discrepância no que toca ao preço de custo dos dois modelos. A solução economicamente mais acessível apenas disponibiliza de 12 portas de entrada/saída, tendo entradas analógicas e digitais mas apenas saídas digitais. Por seu lado, o modelo mais dispendioso disponibiliza de um total de 46 portas de entrada/saída, tendo disponíveis entradas e saídas tanto analógicas como digitais. Em relação a memória de programa disponível, o modelo mais acessível dispõe de apenas 30kB enquanto o modelo mais dispendioso dispõe de 192kB. Em relação aos tipos de linguagem de programação aceites, o modelo mais económico apenas aceita 3 linguagens de programação - *Function Block Diagram*, *Ladder* e *Grafcet*, enquanto a solução mais dispendiosa aceita as 5 linguagens de programação normalizadas. Pelo facto do modelo mais dispendioso ter maior número de portas de entrada/saída, as duas dimensões são mais elevadas quando comparando com o outro modelo.

Tabela 4 - Modelos selecionados da Siemens

	CPU 1211C	CPU314C-2PN/DP
Dimensões (cm)		
Largura	9	12,5
Comprimento	10	12
Profundidade	7,5	13
Tensões		
Alimentação	24 V DC	24 V DC
Entrada Digital	24 V DC	24 V DC
Saída Digital	250 V AC / 30 V DC	250 V AC / 30 V DC
Entrada Analógica	0 - 10 V	0 - 10 V
Saída Analógica	-	0 - 10 V
Corrente (A)		
Saída Digital	2A	2A
Saída Analógica	-	55 mA
Portas		
Entradas Digitais	6	24
Saídas Digitais	4	16
Entradas Analógicas	2	4
Saídas Analógicas	-	2
Tipo de Saída	Relé	Analógica/Relé
Entradas/Saídas Máximas	12	-
Memória (kB)		
Programa	30	192
Dados	10	20
Temperatura (°C)		
Máxima de Funcionamento	60	60
Mínima de Funcionamento	-20	-20
Programação		
Tipos de Linguagem	*Function Block Diagram, Ladder e Grafcet*	*Instruction List, Structured Text, Function Block Diagram, Ladder e Grafcet*
Porta de Programação	Ethernet	RS485/Ethernet
Software Próprio	Sim	Sim
Preço (€)		
Valor no RS	148,00	2041,85

2.5.2. *Software* de Programação

A Siemens dispõe de um *Software* de Programação próprio, chamado Simatic Step 7. Este *Software* de Programação permite um fácil acesso a todos os Autómatos Programáveis da Siemens. Devido à grande variedade de editores de programas, possibilita igualmente a programação nas 5 linguagens de programação de Autómatos. Este *software* engloba uma vasta

gama de funções de fácil acesso, aumentando significativamente a eficiência em todas as tarefas a serem realizadas pelo Autómato.

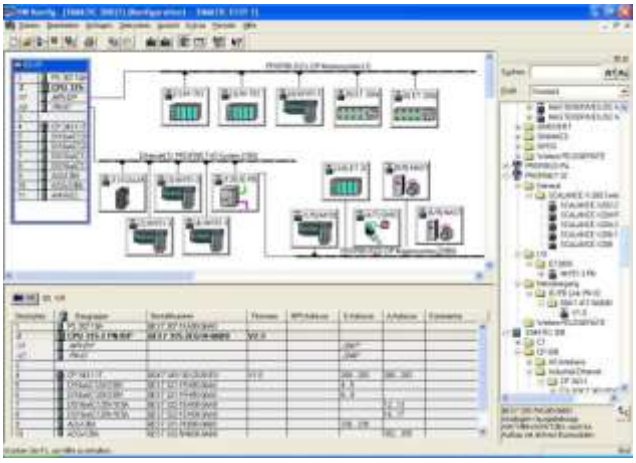


Figura 21 - Simatic Step 7 [40]

As tarefas possíveis passam pela configuração do *hardware*, estabelecimento de comunicações industriais, programação, testes, funções operacionais e diagnóstico [41].

Capítulo 3

3. Fundamentos Teóricos

Neste capítulo serão apresentados os conceitos teóricos necessários para a correta compreensão dos temas ao longo deste trabalho.

3.1 Estrutura dos Autómatos Programáveis

A estrutura dos autómatos programáveis é independente do seu tamanho, partilhando todos os componentes básicos apresentados na figura abaixo.

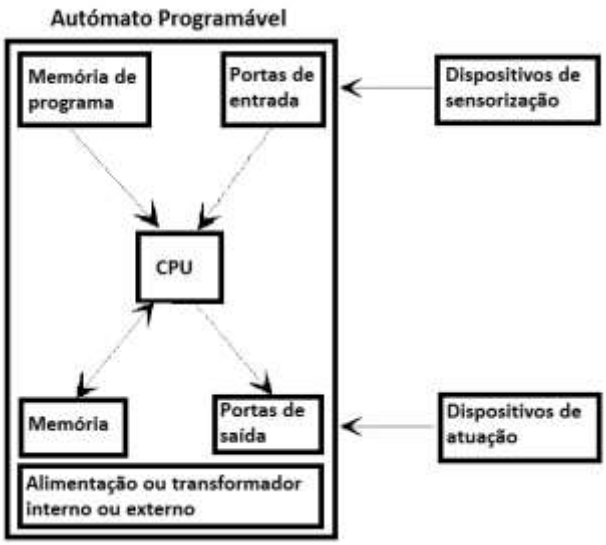


Figura 22 – Estrutura dos autómatos programáveis

O autómato programável é composto por uma Unidade Central de Processamento (CPU) que é responsável pela execução do programa armazenado na memória de programa. No decorrer da execução do programa, sempre que necessário, as portas de entrada são analisadas e as portas de saída são atuadas para comandar o processo, da forma desejada.

Na execução do programa é também utilizada a memória do autómato para armazenamento e transferência de dados. A funcionalidade de armazenamento dos programas, inclusivamente com a alimentação desligada, guarda o estado de todo o processo e recomeçando precisamente nesse mesmo estado, logo que reposta a energia [42].

3.1.1. CPU

O CPU consiste no elemento central do autómato programável e é responsável pela execução do programa criado pelo utilizador. As tarefas da CPU são, para além de executar o programa do utilizador, vigiar o tempo de execução do programa verificando assim se a execução não ficou encravada (*Watch Dog*). Também verifica o estado das entradas e renova o estado das saídas no decorrer da execução do programa [7].

De referir que o transporte da informação entre a memória, portas de entrada e portas de saída para a CPU e vice-versa ocorre com recurso a um barramento de dados [42].

3.1.2. Sistema de Entradas e Saídas

O sistema de entradas e saídas permite a ligação dos dispositivos de sensorização e de atuação, respetivamente, disponibilizando uma análise das entradas e uma atuação nas saídas. Este sistema permite igualmente a interação com o processo e com equipamentos industriais tornando-o num dos componentes mais importantes no autómato programável [42].

O autómato programável, usando os sensores apropriados para o processo a automatizar, pode medir quantidades físicas como: pressão, deslocamento, posição, temperatura, força, deformação, vibração, aceleração, entre outros.

Com recurso ao programa de controlo, o autómato programável controla as saídas, atuando em dispositivos como: motores, cilindros hidráulicos, cilindros pneumáticos, válvulas, alarmes, etc.

O sistema de entradas e saídas pode ser digital, baseado em variáveis binárias, possuindo dois estados (ligado ou desligado/1 ou 0/tudo ou nada); pode ser analógico, assumindo um determinado valor dentro de uma gama especificada pelo fabricante (baseado em conversores A/D e D/A); ou pode ser uma junção de ambas as anteriores, disponibilizando entradas e saídas tanto digitais como analógicas [7].

3.1.3. Memória

Os autómatos programáveis dispõem de duas memórias: a memória de programa, onde é armazenado o programa a ser executado pelo CPU, e a memória do autómato. Esta última é dividida em duas: a memória de dados, onde são armazenados os dados (temporizadores, contadores, variáveis de entrada e saída, entre outros), e a memória do sistema, onde é armazenado o programa que monitoriza o sistema (*Firmware*) [7].

Os tipos de memória atualmente utilizados nos autómatos programáveis são: a RAM, utilizada na fase de desenvolvimento e testes e a EPROM e EEPROM, utilizadas para o armazenamento das configurações do sistema e do programa do processo em código executável [42].

As capacidades de memória variam de autómato para autómato, variando no número máximo de instruções dum programa e na capacidade de dados em memória. Alguns autómatos permitem a utilização de módulos para expansão de memória.

3.2 Linguagens de Programação

A programação dos autómatos é feita com recurso a *softwares* de programação a partir de um computador. Cada *software* de programação pode possibilitar o uso de uma ou mais linguagens de programação, dependendo das linguagens utilizadas pelo fabricante do autómato a utilizar.

A norma IEC 61131-3, já referida no capítulo 1, estabelece as linguagens de programação definindo assim duas linguagens gráficas, duas linguagens de padrão textual e uma linguagem de organização de elementos para programação sequencial. Destas 5 linguagens, a linguagem *Ladder* e *Grafcet* serão detalhadamente explicadas.

3.2.1. Linguagem *Ladder*

A linguagem *Ladder* permite usar diagramas de relés de modo a criar projetos a serem programados em autómatos industriais. Esta linguagem é a mais utilizada pelos fabricantes de autómatos, estando incorporada nos seus *softwares* de programação.

A linguagem consiste num diagrama formado por uma lista de instruções simbólicas interligadas entre si de uma determinada forma. Esta linguagem é composta por 6 categorias de instruções: do tipo relé, de temporização e contagem, de manipulação de dados, aritméticas, de transferência de dados e de controlo de programa.

Um programa em *Ladder* consiste na junção de todos os degraus do projeto, em que cada degrau representa uma equação booleana. Cada degrau controla as saídas a partir de condições de entrada. As entradas e saídas podem ser físicas ou posições internas de memória.

Na figura 23 observa-se a estrutura de um degrau, em que a saída apenas é atuada caso exista continuidade lógica, ou seja, caso o caminho seja fechado desde o início da ramificação até ao fim.

Figura 23 - Degrau *Ladder Diagram*

A posição dos contactos nos degraus depende do controlo lógico desejado. Os contactos podem então ser dispostos em série, em paralelo ou ambos, dependendo do controlo desejado para ativar a saída [42].

3.2.1.1. Instruções do Tipo Relé

As instruções do tipo relé permitem avaliar o estado de uma entrada, de uma saída ou de um ponto interno de entrada ou saída [42].

Início de Ramificação

Dentro de cada degrau é necessário iniciar cada um dos ramos por ele constituintes. O Início de Ramificação designa, então, a primeira instrução quando se pretende realizar alguma equação lógica, apresentando-se o mais à direita possível como se pode observar na Figura 23.

Fim de Ramificação

Em cada degrau é necessário terminar todos os ramos que a constituem a equação lógica, paralelos ou únicos. Assim, Fim de Ramificação designa, então, o fim do caminho da continuidade lógica de um degrau, apresentando-se o mais à esquerda possível como se pode observar na Figura 23.

Contato Normalmente Aberto

Um contacto referencia sempre um endereço que pode ser de uma entrada, um ponto interno ou mesmo uma saída. Se o estado for ligado, o contacto fecha-se e assegura a continuidade lógica, caso contrário, se o estado for desligado, o contacto abre-se deixando de existir continuidade lógica. Na Figura 24 pode observar-se um contacto normalmente aberto.

Figura 24 - Contacto normalmente aberto

Contato Normalmente Fechado

Com um princípio de funcionamento idêntico ao anterior, mas com uma lógica inversa, o contacto normalmente fechado abre-se, quando o estado for ligado, deixando de existir continuidade lógica. Caso contrário, se o estado for desligado, o contacto fecha-se assegurando a continuidade lógica. Na Figura 25 observa-se um contacto normalmente fechado.

Figura 25 - Contacto normalmente fechado

Saída Normalmente Aberta

Uma saída referencia sempre um endereço que pode ser um ponto interno ou uma saída. Em caso de existir continuidade lógica o ponto interno ou saída é ligado e caso não exista a continuidade lógica completa estes são desligados. Na Figura 26 pode observar-se uma saída normalmente aberta.

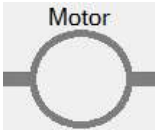


Figura 26 - Saída normalmente aberta

Saída Normalmente Fechada

Com um princípio de funcionamento idêntico ao anterior, mas com uma lógica inversa, em caso de existir continuidade lógica o ponto interno ou saída é desligado e caso não exista a continuidade lógica completa estes são ligados. Na Figura 27 observa-se uma saída normalmente fechada.

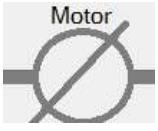


Figura 27 - Saída normalmente fechado

Exemplos

Na Figura 28 observa-se um degrau em linguagem *Ladder*, da equação booleana $(Start + Motor) * \overline{Stop} = Motor$.

Figura 28 - Exemplo 1 de um degrau em *Ladder Diagram*

Alterando a saída de normalmente aberta para normalmente fechada a equação booleana passa a ser $\overline{(Start + Motor) * \overline{Stop}} = Motor$. O degrau resultante pode, então, observar-se na Figura 29.

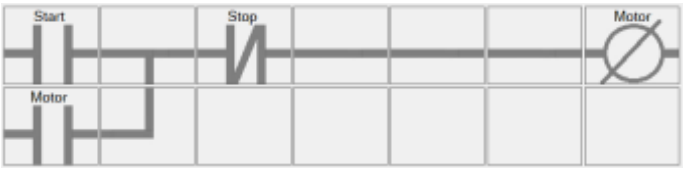


Figura 29 - Exemplo 2 de um degrau em *Ladder Diagram*

Com estes dois exemplos pode verificar-se que um "OR" lógico corresponde a uma ligação em paralelo entre símbolos, enquanto uma negação lógica corresponde à utilização de contactos ou saídas normalmente fechadas.

Analisando detalhadamente o primeiro exemplo, Figura 28, o acionamento e paragem do motor é feito com recurso às tradicionais botoneiras "Start" e "Stop". De referir que para manter o Motor em funcionamento após largar o botão de "Start" é necessário fazer a autoalimentação. Premindo o botão "Start" e não estando o botão "Stop" ativo, passa a existir continuidade lógica, o Motor arranca e o contacto do Motor é ativado. Assim, mesmo que se solte o botão "Start" a continuidade lógica passa a ser assegurada pelo contacto do Motor.

Premindo o botão "Stop" passa a não existir continuidade lógica, o Motor é parado e o contacto do Motor deixa de estar acionado. Neste caso mesmo que o botão "Stop" deixe de ser premido, continua a não existir continuidade lógica pois o contacto "Start" e o contacto Motor estão abertos.

3.2.1.2. Instruções de temporização e contagem

As instruções de temporização e contagem consistem em instruções de saída com funções semelhantes às dos temporizadores e contadores mecânicos ou eletrónicos. Usualmente estes são utilizados para ligar um dispositivo, ou mesmo desligar, ao fim de um determinado tempo ou

contagem. O princípio de funcionamento é semelhante pois ambos têm a funcionalidade de contador. O temporizador conta os instantes de tempo necessários para a duração pretendida, enquanto o contador regista o número de ocorrências.

As duas instruções, tanto de temporização como de contagem, utilizam dois registos: um para armazenar o número de contagens e outro para armazenar o valor inicial.

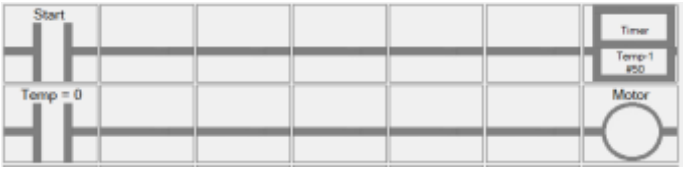


Figura 30 - Exemplo de um degrau com um temporizador em *Ladder Diagram*

Na Figura 30 observa-se um temporizador que é iniciado quando é premido o botão "Start". Este temporizador irá atuar na saída Motor ao fim de 50 impulsos de relógio. Pode-se também observar que o primeiro registo armazena o número de contagens já efetuadas e o segundo registo armazena o valor desejado para atuar o Motor.

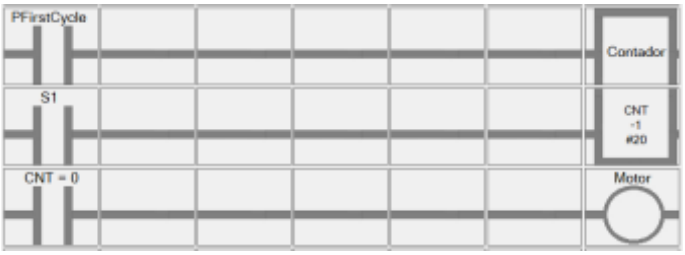


Figura 31 - Exemplo de um degrau com um contador em *Ladder Diagram*

Na Figura 31 observa-se um contador que é iniciado após no primeiro segundo de execução do programa. Este contador permite registar eventos externos no primeiro registo e no segundo registo o valor desejado para atuar a saída. Quando existirem 20 ocorrências do evento externo o contador atua ativando a saída.

3.2.2. Linguagem *Grafcet*

A linguagem *Grafcet* consiste na representação gráfica de algoritmos usados em sistemas de controlo. Esta linguagem é considerada uma linguagem de boa legibilidade devido à utilização de símbolos gráficos na representação de algoritmos.

O diagrama *Grafcet* engloba três entidades: etapas, condições de transição e tarefas. Após uma etapa e caso cumpra as condições nas transições, o automatismo fica sequencial ativando as tarefas nas seguintes etapas. Em cada etapa são ativas ações ou tarefas (conjunto de ações), onde

são acionadas as saídas do processo, ficando estas últimas ativas até sair da etapa (transição seguinte ser validade) [42].

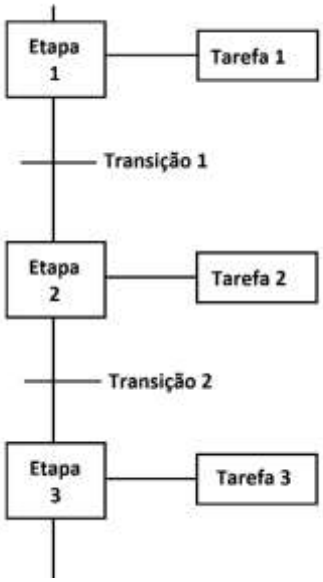


Figura 32 - Etapas, transições e tarefas de um *Grafcet*

Na Figura 32 pode observar-se um exemplo genérico de *Grafcet*. Enquanto a etapa 1 estiver ativa a tarefa 1 encontra-se igualmente ativa. Para que a etapa 2 fique ativa é sempre necessário que a etapa 1 esteja ativa e que seja respeitada a transição 1, deixando a etapa 1 e as suas respetivas tarefas de estarem ativas. Para o processo ser sequencial é necessária a progressão entre etapas.

Sendo esta linguagem uma ferramenta de estruturação de programas ela não substitui as restantes linguagens.

3.2.2.1. Estrutura Lógica *OR*

A estrutura lógica *OR,* como o próprio nome indica, utiliza o operador lógico *OR*, também denominado por "ou". Este tipo de estrutura é usado exclusivamente em sequências alternativas criando um processo sequencial caso, pelo menos, umas das transições usadas na linguagem, seja verificada. Na Figura 33 pode observar-se um exemplo genérico da estrutura lógica *OR*, incorporando a divergência e convergência *OR* em que, estando a etapa 1 ativa existem duas alternativas e para a etapa 4 ficar ativa existem também duas alternativas.

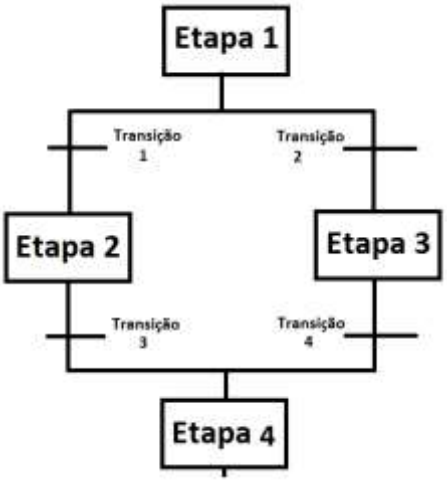


Figura 33 - Estrutura lógica *OR*

Divergência OR

A divergência *OR* permite sequências alternativas, quando o automatismo ou processo assim o exige. Na Figura 34 observa-se que, caso esteja a etapa 1 ativa, existem duas alternativas: a transição 1 é validada e passa a etapa 2 a estar ativa ou a transição 2 é validada e passa a etapa 3 a estar ativa.

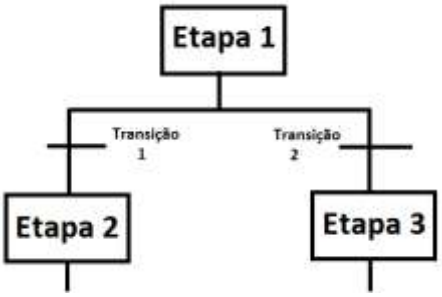


Figura 34 - Divergência *OR*

A Tabela 5 foi elaborada com base na Figura 34 observando-se que, caso esteja a etapa 1 ativa, existem duas alternativas, passar a estar ativa a etapa 2 ou etapa 3.

Tabela 5 - Divergência *OR*

Transição 1	**Transição 2**	**Resultado**
Não Validada (0)	Não Validada (0)	Etapa 1 – ativa
Validada (1)	Não Validada (0)	Etapa 2 – ativa
Não Validada (0)	Validada (1)	Etapa 3 – ativa
Validada (1)	Validada (1)	Caso não possível

Convergência OR

A convergência *OR* utiliza-se no caso de execução simultânea de sequências que chegam a uma mesma etapa. Na Figura 35 observa-se que para a etapa 4 estar ativa existem duas alternativas: a etapa 2 estar ativa e a transição 3 seja validada, ou a etapa 3 estar ativa e a transição 4 seja validada.

Figura 35 – Convergência *OR*

A Tabela 6 foi elaborada com base na Figura 35 observando-se que, para etapa 4 estar ativa, existem duas alternativas, vindo da etapa 2 ou vindo da etapa 3.

Tabela 6 - Convergência *OR*

Transição 3	**Transição 4**	**Resultado**
Não Validada (0)	Não Validada (0)	Etapa 2 ou 3 – ativa
Validada (1)	Não Validada (0)	Etapa 4 – ativa (caso a Etapa 2 estivesse ativa)
Não Validada (0)	Validada (1)	Etapa 4 – ativa (caso a Etapa 3 estivesse ativa)
Validada (1)	Validada (1)	Caso não possível

3.2.2.2. Estrutura Lógica *AND*

A estrutura lógica *AND*, como o próprio nome indica, utiliza o operador lógico *AND*, também denominado por "e". Este tipo de estrutura é usada exclusivamente para ativar sequências paralelas a serem realizadas simultaneamente.

Na Figura 36 pode observar-se um exemplo genérico da estrutura lógica *AND*, incorporando a divergência e convergência *AND* em que, estando a etapa 1 ativa e a transição 1 validada, são realizadas simultaneamente duas sequências paralelas e, para que a etapa 6 fique ativa, é necessário que ambas as sequências paralelas terminem.

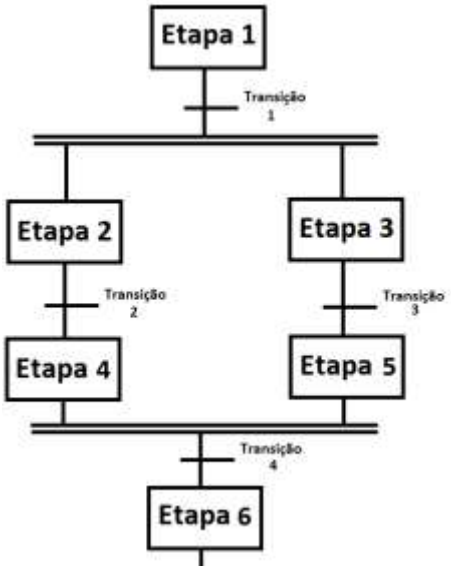


Figura 36 - Estrutura lógica *AND*

Divergência *AND*

A divergência *AND* utiliza-se caso o processo ou automatismo exija ativar simultaneamente sequências paralelas. Na Figura 37 observa-se que, se a etapa 1 estiver ativa e a transição 1 validada, as etapas 2 e 3 ficam ativas e a etapa 1 deixa, então, de estar ativa.

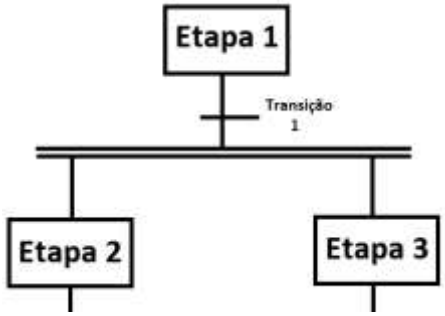


Figura 37 - Divergência *AND*

A Tabela 7 foi elaborada com base na Figura 37 observando-se que, estando a etapa 1 ativa e a transição 1 validada, as etapas 2 e 3 ficam ativas.

Tabela 7 - Divergência *AND*

Transição 1	**Resultado**
Não Validada (0)	Etapa 1 – ativa
Validada (1)	Etapa 2 e 3 – ativa

Convergência *AND*

A convergência *AND* utiliza-se quando se dá o caso de terminarem sequências simultaneamente paralelas, convergindo-as simultaneamente para o resto do processo ou automatismo.

Na Figura 38 observa-se que, para etapa 6 estar ativa, é necessário que ambas as sequências paralelas terminem, ou seja, só quando a sequência que acaba na etapa 4 e a sequência que acaba na etapa 5 são terminadas e estando validada a transição 4 é que a etapa 4 passa a estar ativa.

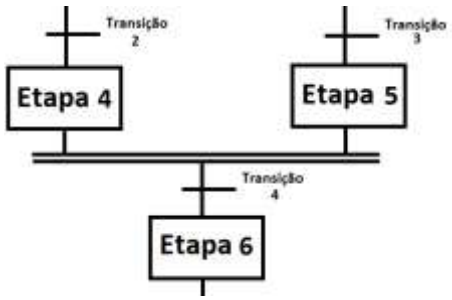


Figura 38 - Convergência *AND*

A Tabela 8 foi elaborada com base na Figura 38 observando-se que, para a etapa 4 estar ativa, é necessário que ambas as sequências paralelas terminem, mesmo considerando que a transição 4 seja validada.

Tabela 8 - Convergência *AND*

Etapa 4	**Etapa 5**	**Resultado**
Não Ativa (0)	Não Ativa (0)	Etapa 6 – não fica ativa mesmo que a transição 4 seja validada
Ativa (1)	Não Ativa (0)	Etapa 6 – não fica ativa mesmo que a transição 4 seja validada
Não Ativa (0)	Ativa (1)	Etapa 6 – não fica ativa mesmo que a transição 4 seja validada
Ativa (1)	Ativa (1)	Etapa 6 – ativa caso a transição 4 seja validada

Capítulo 4

4. Estrutura do Sistema

Neste capítulo será apresentada a estrutura desenvolvida para a implementação da solução de *hardware* e *software* com o propósito de atingir os objetivos propostos.

4.1 Introdução

A estrutura escolhida baseia-se em dois pontos: a estrutura dos autómatos programáveis e a estrutura do ambiente de desenvolvimento multilinguagem, como já foi referido no capítulo dos fundamentos teóricos. Esta estrutura encontra-se então dividida em duas partes: *hardware* de um autómato programável e software de programação do respetivo autómato.

A primeira parte, *hardware*, é constituída pelos três pontos da estrutura dos autómatos programáveis, mencionados no capítulo dos fundamentos teóricos: CPU, sistema de entradas/saídas e memória. Foi escolhido um microprocessador *Arduino* que, com dimensões reduzidas, torna possível ter um CPU para a execução do programa criado pelo utilizador, verificando a cada ciclo o estado das entradas e renovando o estado das saídas, dispondo de um sistema de entradas é possível a utilização dos ADC disponibilizados.

O microprocessador *Arduino* dispõem de memória de programa, onde é armazenado o programa a ser executado pelo CPU e de memória de dados, onde são armazenados os dados (temporizadores, contadores, variáveis de entrada e saída, entre outros). As saídas deste microprocessador são saídas digitais em 0-5v pelo que foram escolhidos relês de estado solido por forma a ligar ou desligar dispositivos de maior potência.

A segunda parte, *software* de programação, é constituída pelos dois pontos das linguagens de programação, mencionados no capítulo dos fundamentos teóricos: a linguagem *Ladder* e a

linguagem *Grafcet*. Foi desenvolvido um ambiente de desenvolvimento multilinguagem, utilizando o Visual Studio 2013 por forma a ser possível a criação do projeto em linguagem *Ladder*, linguagem *Grafcet* e linguagem C, com possibilidade de depuração de erros das respetivas linguagens e posterior programação do autómato.

Esta introdução pretendeu dar a conhecer a estrutura de todo o sistema desenvolvido. De seguida serão apresentados, em detalhe, os sistemas desenvolvidos, tanto para a componente de *hardware* como para a componente de software.

4.2 Implementação Prática

A implementação da estrutura foi elaborada após a escolha das melhores soluções de forma a satisfazer os requisitos pré-definidos. Na Figura 39 é demostrado o sistema físico desenvolvido. Pode então observar-se o computador portátil, onde é corrido o ambiente de desenvolvimento, e o autómato programável, onde é corrido o projeto criado. A ligação entre eles é feita através de um cabo USB-micro USB. Pode ainda observar-se que a alimentação do autómato é feita através do cabo USB.

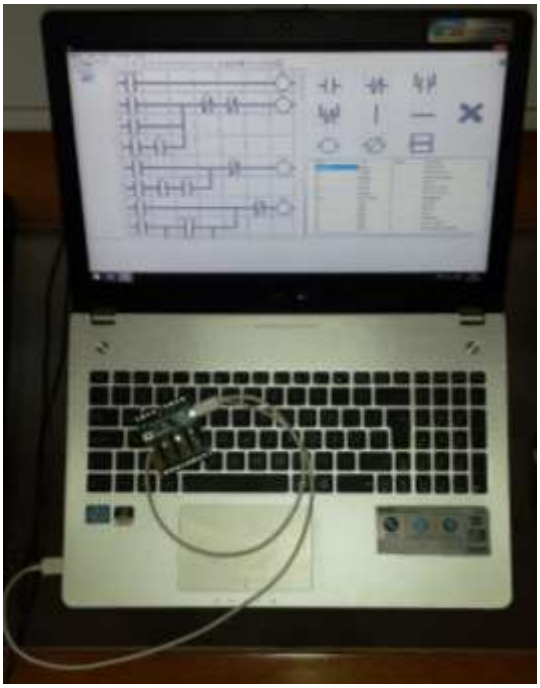


Figura 39 - Sistema físico implementado

4.3 *Hardware* Desenvolvido

O *hardware* desenvolvido está divido em duas partes: num primeiro protótipo, que apresentou algumas limitações, e num segundo protótipo, o protótipo final. Serão então apresentados os dois sistemas desenvolvidos bem como as limitações do primeiro.

4.3.1. Primeiro Protótipo

Neste primeiro protótipo a alimentação era feita através da utilização de um transformador 230V-12V externo ao autómato e de um circuito integrado que regula a para 5V, LM7805, interno ao autómato. O microprocessador utilizado era o ATMEGA 328P com recurso a um relógio, ou oscilador, externo de 16 MHz e ao programador *Arduino* USB *Serial Light Adapter*. O esquema elétrico do primeiro protótipo encontra-se apresentado na Figura 40.

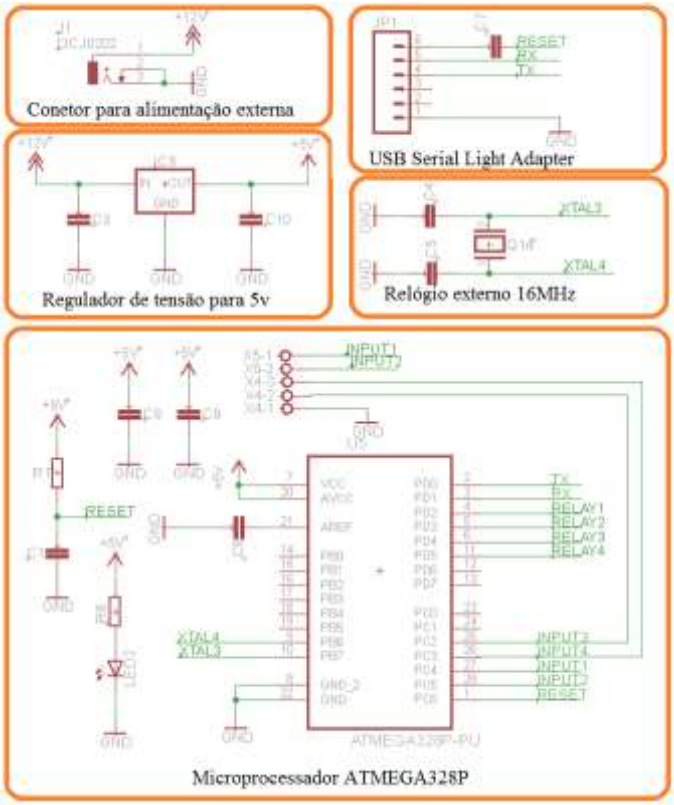


Figura 40 - Esquema elétrico do primeiro protótipo – parte do microprocessador

4.3.1.1. Regulador de Tensão

O regulador de tensão escolhido foi o LM7805, Figura 41, que consiste num circuito integrado que regula a tensão para 5V podendo fornecer uma corrente de saída até a 1.5A. Assim sendo, ligando o transformador 230V-12V ao conector para alimentação externa, o regulador de tensão coloca na sua saída 5V para alimentar os restantes componentes do esquema elétrico.

Figura 41 - Regulador de Tensão LM7805

4.3.1.2. Programador *Arduino* USB *Serial Light Adapter*

O programador *Arduino* USB *Serial Light Adapter* converte uma conexão USB numa conexão serie, com RX e TX. De um lado o conetor mini-USB que, através do cabo, permite a ligação a um computador. No outro lado, os 5 pinos permitem a ligação a um microprocessador. Estes 5 pinos são: RX, que permite receber dados do computador; TX, que permite enviar dados para o computador; +5V, que permite alimentar circuitos através da alimentação da porta USB do computador; GND, a massa do circuito que é ligada a massa do circuito de alimentação externa; e um sinal de *Reset* que apenas é usado no momento da programação. Pode então observar-se o programador na Figura 42.

Figura 42 - *Arduino* USB *Serial Light Adapter*

4.3.1.3. ATMEGA 328P

O microprocessador escolhido foi o ATMEGA 328P que dispõe de 32 pinos, embora apenas 23 destes possam ser usados como pinos de entrada/saída. Dispõem igualmente de dois temporizadores de 8-bits, um de 16-bits, seis canais de PWM e seis canais ADC de 10-bits de resolução. É um microprocessador de 8-bits que funciona com um relógio externo de 16MHz e dispõe de 32Kbytes de memória *flash*. Na Figura 43 pode observar-se o microprocessador na respetiva placa de circuito impresso.

Figura 43 - ATMEGA 328P

4.3.1.4. Sistema de saídas

As saídas deste microprocessador são digitais, ou seja, entre 0-5V, pelo que foram escolhidos relês de estado sólido por forma ligar ou desligar dispositivos de maior potência. Os relês de estado sólido têm um funcionamento semelhante a relês mecânicos, recorrendo a contatos óticos em vez de contactos mecânicos. Assim, podem ser ligados mais rapidamente e apresentam tempos de vida mais longos não existindo desgaste.

Devido à corrente de saída dos pinos do microprocessador ser até 20 mA foi necessária a colação de um transístor NPN, tendo sido escolhido o 2N2222. Na Figura 44 pode observar-se o esquema das saídas em que cada saída tem um transístor e um relê de estado sólido.

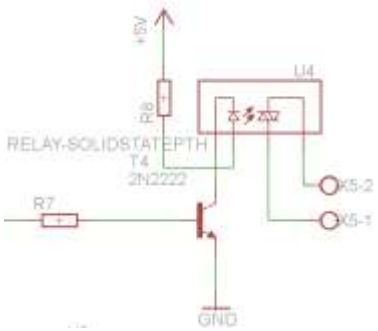


Figura 44 - Esquema elétrico do primeiro protótipo – parte das saídas

Relê de Estado Sólido

O relê de estado sólido escolhido foi o modelo S108T02 da Sharp devido às suas características e pelo facto de ser de dimensões reduzidas. O princípio de funcionamento de um relê de estado sólido é baseado num LED de infravermelhos que, quando ligado, estimula um *Phototriac* que fecha o circuito. O isolamento elétrico é garantido, como nos relês mecânicos, ocorrendo, neste caso, através de foto acoplamento. O relê de estado sólido escolhido é apresentado na Figura 45 podendo comutar 400 V AC e 8A.

Figura 45 - Relê de estado sólido Sharp

4.3.1.5. Placa de circuito impresso

Com recurso ao *software Eagle* foi possível, após a criação do esquema elétrico, a criação da placa de circuito impresso. Na Figura 46 pode observar-se a placa de circuito impresso do primeiro protótipo, sendo uma placa de face única com recurso a cinco ligações, feitas na fase superior, por forma a reduzir o custo da placa.

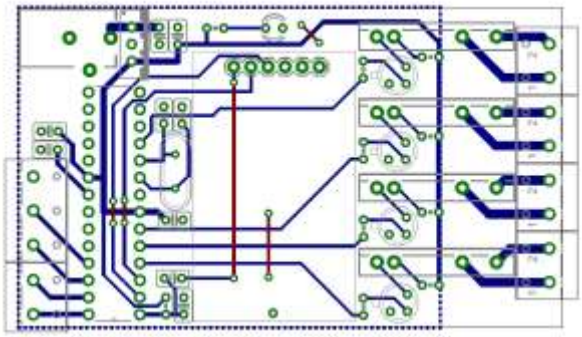


Figura 46 - Primeiro protótipo

4.3.1.6. Limitações Encontradas

Neste primeiro protótipo foram encontradas duas limitações. A primeira prende-se com a dificuldade em programar o respetivo microprocessador devido ao programador não realizar o *Reset* automático no momento de programação. Foi verificado que o programador conseguia estabelecer a comunicação com o computador e, através da ligação entre o sinal RX e o sinal TX, foi possível verificar que os caracteres enviados pelo computador eram novamente recebidos, embora, no momento de programação, fosse exibido um erro por falta do *Reset* ao microprocessador.

A segunda limitação encontrada tem a ver com necessidade de ligação da alimentação externa, mesmo no momento de programação, o que transformaria este protótipo numa solução pouco prática e de difícil integração.

Assim, como teremos oportunidade de observar de seguida, o segundo protótipo desenvolvido teve em consideração as limitações registadas anteriormente. Foram também acrescentadas duas saídas, uma para alimentação de sensores e outra para a comunicação com outros autómatos.

4.3.2. Segundo Protótipo

Neste segundo protótipo a alimentação é providenciada pela porta USB do computador portátil ou através da ligação a um transformador com saída USB, 230V-5V, externo. O microprocessador utilizado é o ATMEGA32u4 inserido na placa *Arduino Micro*, não necessitando de um programador

externo, contrariamente ao primeiro protótipo. O esquema elétrico do segundo protótipo encontra-se apresentado na Figura 47. Como se pode constatar, o sistema de saídas não sofreu nenhuma alteração em relação ao primeiro protótipo.

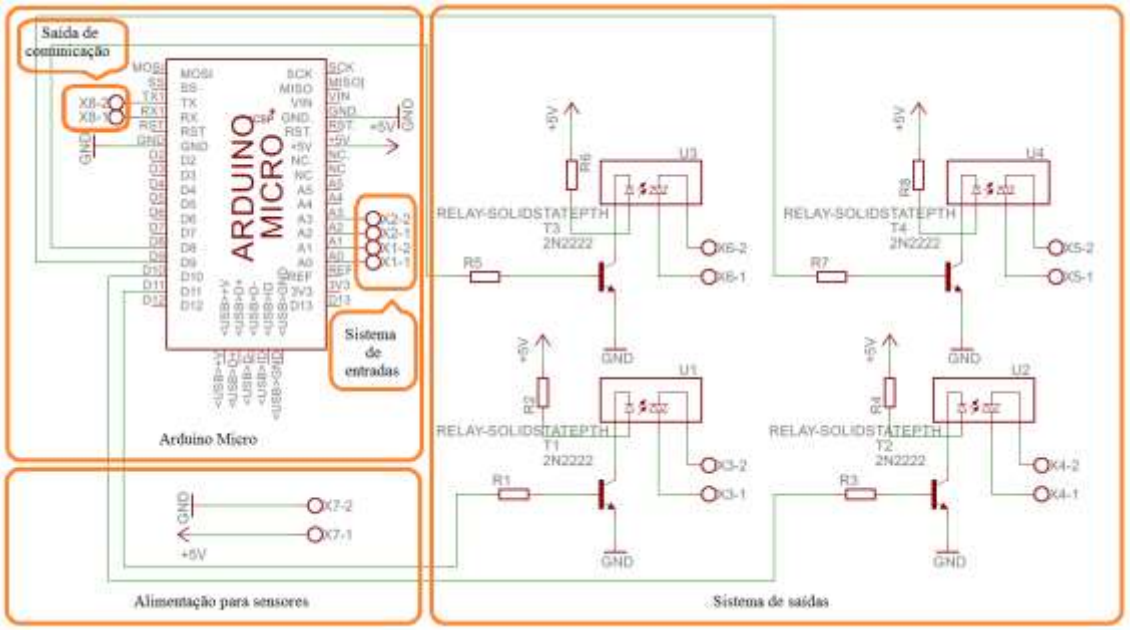


Figura 47 - Esquema elétrico do segundo protótipo

4.3.2.1. *Arduino Micro*

O *Arduino Micro* é um microcontrolador que utiliza o microprocessador ATMEGA32u4, dispondo de um relógio, ou oscilador, de 16 MHz, uma ligação ICSP, uma conexão micro USB e um botão de *Reset*. Este microcontrolador apesar das suas dimensões reduzidas contém as funcionalidades que um microcontrolador deve conter, bastando conectá-lo a um computador com um cabo USB. Na Figura 48 pode observar-se o microcontrolador *Arduino Micro*.

Figura 48 - *Arduino Micro*

ATMEGA32u4

O microprocessador inserido no microcontrolador *Arduino Micro* é o ATMEGA32u4. Dispondo de 44 pinos mas apenas 26 destes podem ser usados como pinos de entrada/saída. Dispõem também de um temporizador de 8-bits, dois temporizadores de 16-bits, quatro canais de PWM de

8-bits, quatro canais de PWM de resolução programada entre 2 a 16 bits e doze canais ADC de 10-bits de resolução com ganho programável. É um microprocessador de 8-bits que funciona com um relógio de 16MHz e dispõe de 32Kbytes de memória *flash*.

4.3.2.2. Placa de circuito impresso

No *software Eagle* após a criação do esquema elétrico, foi possível a criação da placa de circuito impresso. Na Figuro 49 observa-se a placa de circuito impresso do segundo protótipo sendo esta uma placa de face única, de menor dimensão do que o primeiro protótipo e com recurso a três ligações, feitas na fase superior, por forma a reduzir o custo da placa.

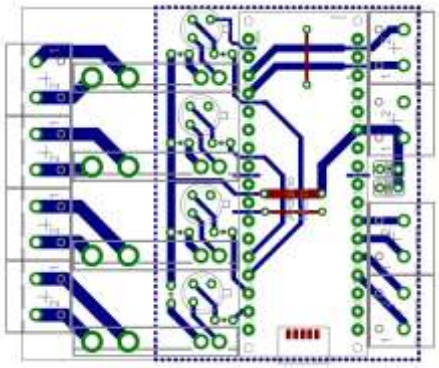


Figura 49 - Segundo protótipo

4.4 *Software* Desenvolvido

Neste subcapítulo, pretende-se descrever os algoritmos desenvolvidos em C# no *software Visual Studio* 2013. Será explicado o ambiente de desenvolvimento multilinguagem, numa fase inicial através de uma visão geral e posteriormente numa visão mais aprofundada, *form* a *form*.

4.4.1. Ambiente de Desenvolvimento Multilinguagem

O ambiente de desenvolvimento multilinguagem foi elaborado de forma a ser o mais simples possível tornando-o de fácil utilização. É constituído por uma janela principal intitulada de *form1* e doze janelas secundárias: a janela de novo ficheiro, *form* 2; a janela sobre, *form*3; a janela adicionar símbolo na linguagem *Grafcet*, *form*4; a janela transição na linguagem *Grafcet*, *form*5; a janela etapas na linguagem *Grafcet*, *form*6; a janela eliminar código na linguagem de *Grafcet*, *form*7; a janela programação, *form*8; a janela de símbolos na linguagem *Ladder*, *form*9; a janela de entradas na linguagem *Ladder*, *form*10; a janela de saídas na linguagem *Ladder*, *form*11; a

janela de saídas com opções diversas, *form*12; e a janela eliminar código em linguagem *Ladder*, *form*13. De seguida, estas treze janelas serão apresentadas mais detalhadamente.

Na Figura 50 poderá observar-se o diagrama UML do ambiente de desenvolvimento, onde é possível verificar que existe um ficheiro intitulado variáveis que contem as variáveis globais de todas as *form's*. Por fim, serão verificadas as treze *form's* bem com os respetivos nomes.

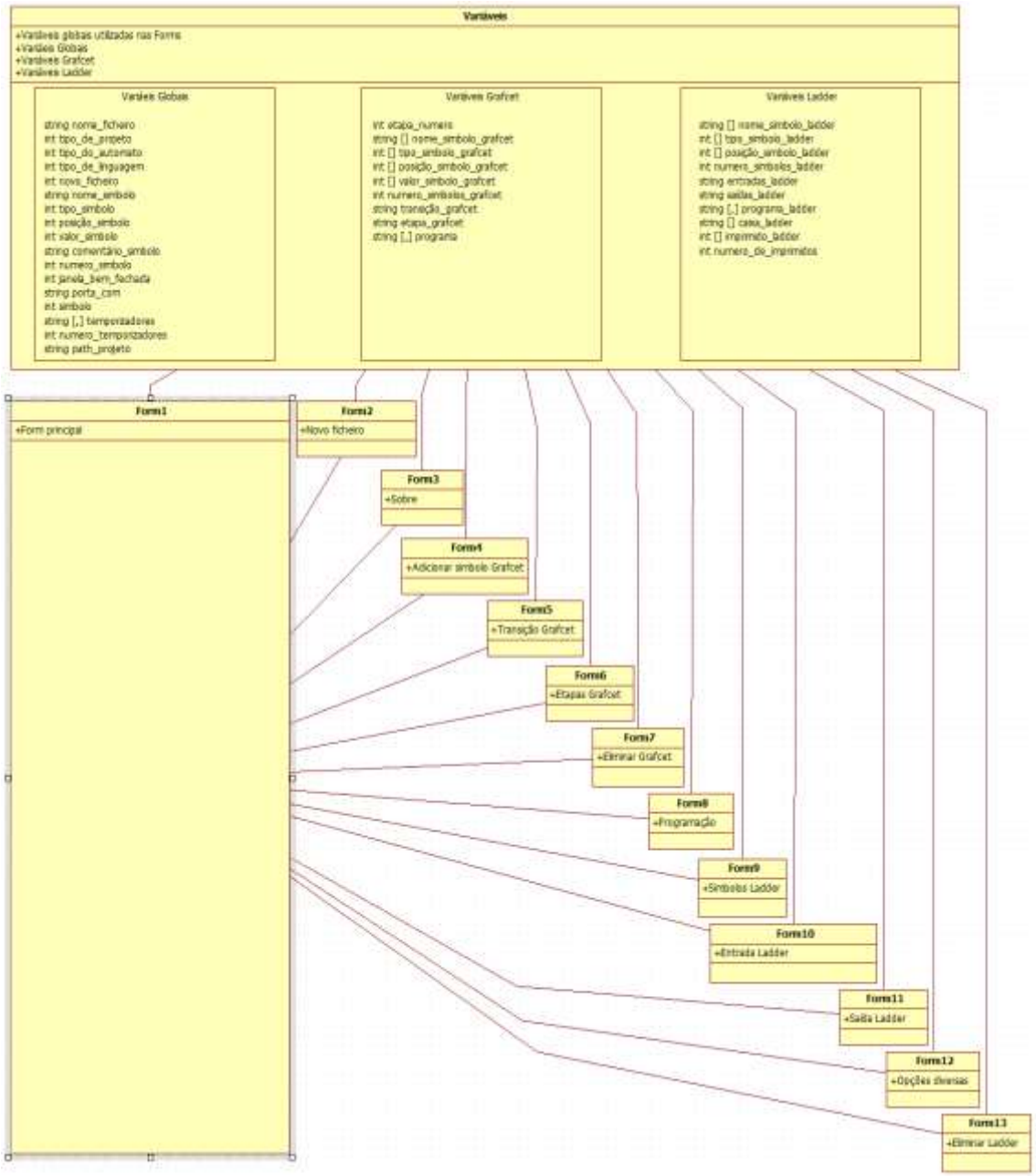


Figura 50 - Diagrama UML do Ambiente de Desenvolvimento Multilinguagem

4.4.1.1. Janela “Principal”

A *Form*1 é a janela “Principal” do ambiente de desenvolvimento multilinguagem. Nela encontram-se as funções principais do projeto assim como as chamadas das janelas secundárias. Na Figura 51 pode observar-se as funções pertencentes a esta *form*.

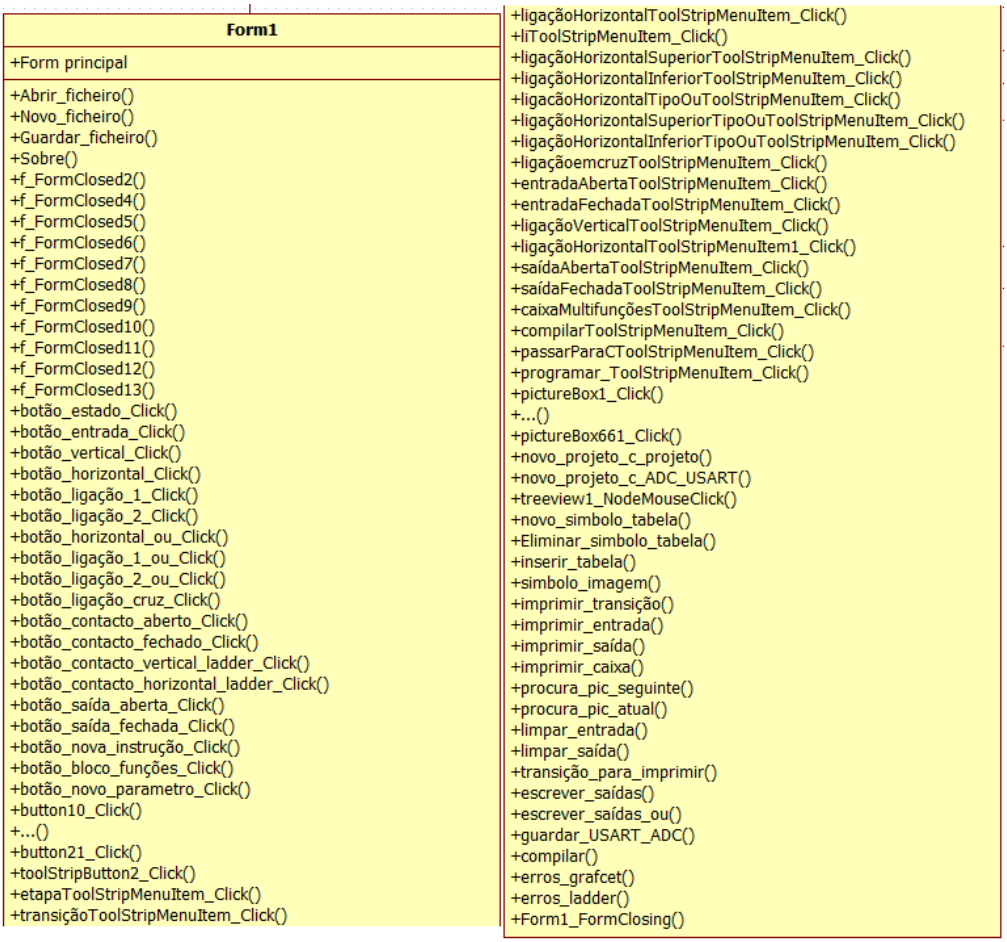


Figura 51 - Funções da *Form*1

A função “abrir_ficheiro()” abre um projeto já existente, num ficheiro de texto, carregando automaticamente a tabela de símbolos e o código na respetiva linguagem. A função “novo_ficheiro()” chama a janela secundária intitulada como *Form*2, janela “Novo Ficheiro”, e quando esta é fechada entra na função “f_FormClosed2()” que inicia um projeto na linguagem escolhida, com o nome inserido e para o tipo de autómato selecionado. A função “guarda_ficheiro()” guarda o projeto que se encontra aberto num ficheiro de texto. A função “sobre()” chama a janela secundária intitulada como Form3, janela “Sobre”, que apresenta as informações do ambiente de desenvolvimento.

No caso de o utilizador escolher como linguagem pretendida a linguagem C, são chamadas as funções "novo_projeto_c_projeto()" e "novo_projeto_c_ADC_USART()" que criam os modelos para a programação em linguagem C, contendo as inicializações e funções para o uso dos canais ADC disponíveis no autómato e da comunicação porta-serie.

Figura 52 - *Form*1, Ambiente de Desenvolvimento

Na Figura 52 pode-se observar a janela "Principal" que é apresentada na abertura do ambiente de desenvolvimento.

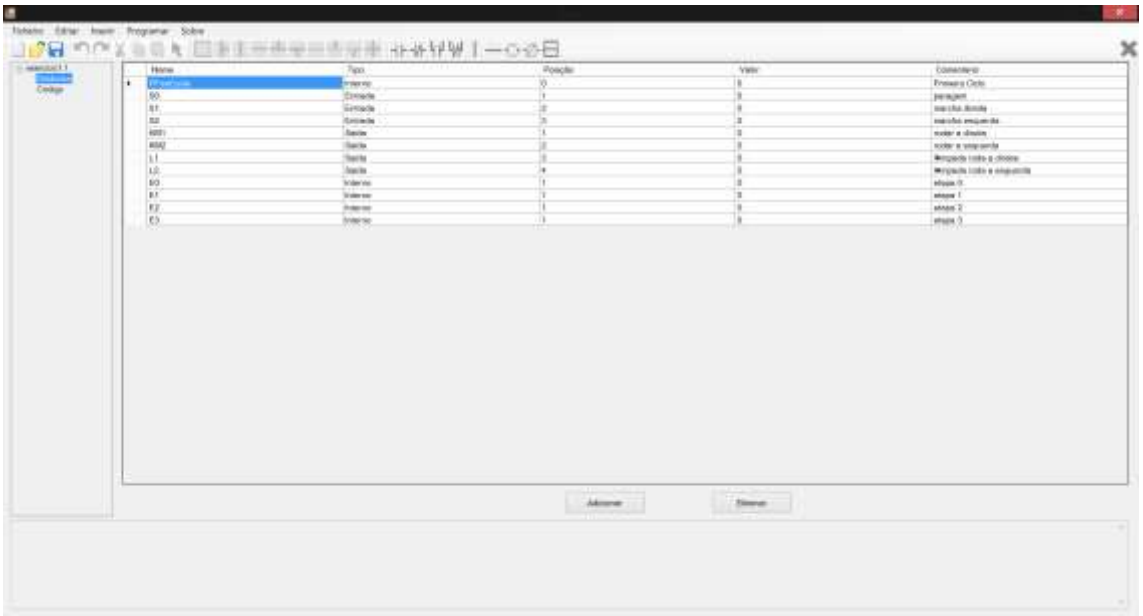


Figura 53 - *Form*1, Janela da tabela de símbolos *Ladder*

Na Figura 53 observa-se a janela que apresenta a tabela de símbolos na linguagem *Ladder*. A função "novo_simbolo_tabela()" é chamada quando é pressionado o botão "Adicionar" e, caso o projeto seja na linguagem *Grafcet,* chama a janela secundária intitulada como *Form*4, janela "Símbolos *Grafcet*". Quando esta janela é fechada entra na função "f_FormClosed4()" e posteriormente na função "inserir_tabela()". Caso o projeto seja na linguagem *Ladder*, chama a janela secundaria intitulada *Form*9, janela "Símbolos *Ladder*", e fechando esta janela, entra na função "f_FormClosed9()" e posteriormente na função "inserir_tabela()".

A função "Eliminar_simbolo_Click()" é chamada quando é pressionado o botão "Eliminar" e elimina o item que estiver selecionado na tabela.

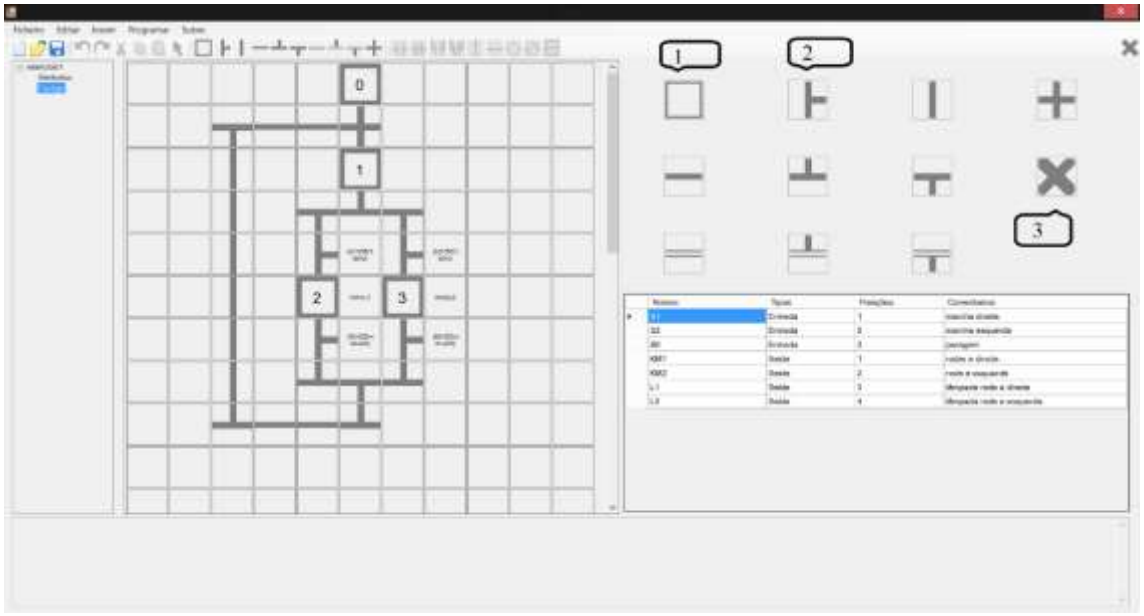


Figura 54 - *Form*1, Janela código *Grafcet*

Na Figura 54 observa-se a janela de código *Grafcet*. Para inserir um constituinte do código é necessário clicar nesse mesmo constituinte e posteriormente na caixa de imagem que se pretende inserir. Os constituintes são disponibilizados à direita da janela. No ambiente de desenvolvimento existe um total de 661 caixas de imagem das quais 276 são usadas para o código em *Grafcet*, ou seja, caso se pressione uma caixa de imagem, de 1 a 276, é chamada a função correspondente de "pictureBox1_Click()" à "pictureBox276_Click()". Em qualquer destas funções é feita a chamada da função "simbolo_imagem()" que insere nessa caixa de imagem o constituinte anteriormente pressionado.

Na Figura 54, caso seja pressionado o botão de etapa que está marcado com o número 1, a variável global "símbolo" fica com o valor 1 e, ao ser pressionada uma caixa de imagem, é

chamada a função "simbolo_imagem()". Esta, chama a janela secundária intitulada como Form6, janela "Etapa *Grafcet*", onde são escolhidas as saídas ativadas por esta etapa. Fechando esta janela, entra na função "f_FormClosed6()" que chama a função "imprimir_etapa()" onde é inserido, na caixa de imagem, o símbolo etapa e, na caixa à sua direita, as saídas ativadas por essa etapa.

Caso seja pressionado o botão de transição, que está marcado com o número 2, a variável global "símbolo" fica com o valor 2 e, após ser pressionada uma caixa de imagem, entra na função "simbolo_imagem()". Esta, chama a janela secundária intitulada como *Form*5, janela "Transição *Grafcet*", onde é escolhida a transição a ser inserida no código. Fechada esta janela, entra na função "f_FormClosed5()" que chama a função "imprimir_transição()" onde é inserido, na caixa de imagem, o símbolo transição e, na caixa à sua direita, a transição.

Caso seja pressionado o botão de "Eliminar", marcado com o número 3, é chamada a janela secundária intitulada como *Form*7, janela "Eliminar *Grafcet*", onde são eliminadas partes constituintes do código *Grafcet*. Fechada esta janela, entra na função "f_FormClosed7()" onde são limpas todas as caixas de imagem e novamente preenchidas com as partes não eliminadas pelo utilizador.

Sendo pressionado qualquer outro botão disponível no painel à direita desta janela, é inserida, na caixa de imagem selecionada, o símbolo previamente pressionado sem abrir uma janela secundária.

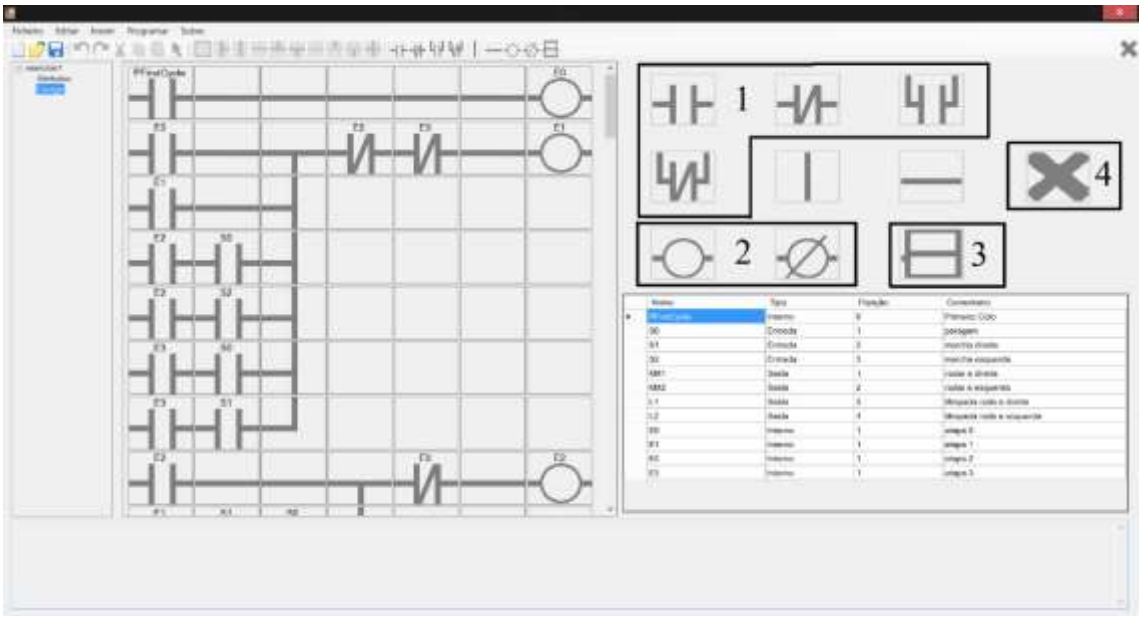


Figura 55 - *Form*1, Janela código *Ladder*

Na Figura 55 observa-se a janela de código *Ladder*, com o mesmo princípio de funcionamento que no código *Grafcet*. Para inserir um constituinte do código é necessário clicar no respetivo constituinte e posteriormente na caixa de imagem que se pretende inserir. Os constituintes são disponibilizados no painel à direita da janela. No ambiente de desenvolvimento existe um total de 661 caixas de imagem das quais 384 são usadas para o código em *Ladder*, ou seja, caso seja pressionada uma caixa de imagem, de 277 a 661, é chamada a função correspondente da "pictureBox277_Click()" à "pictureBox661_Click()". Em qualquer destas funções é feita a chamada da função "simbolo_imagem()" que insere nessa caixa de imagem o constituinte anteriormente pressionado.

Na Figura 55 pode observar-se, caso seja pressionado qualquer botão que esteja marcado na zona com o número 1, e ao ser pressionada uma caixa de imagem, a chamada da função "simbolo_imagem()". Esta, chama a janela secundária intitulada como *Form*10, janela "Entrada *Ladder*", onde é escolhida a entrada a ser inserida. Fechada esta janela entra na função "f_FormClosed10()" que chama a função "imprimir_entrada()" onde é inserido, na caixa de imagem, o símbolo do constituinte que foi previamente pressionado e a entrada escolhida pelo utilizador.

Caso seja pressionado um dos dois botões que estejam marcados na zona com o número 2, e ao ser pressionada uma caixa de imagem, entra na função "simbolo_imagem()". Esta, chama a janela secundária intitulada como *Form*11, janela "Saída *Ladder*", onde é escolhida a saída a ser inserida. Fechada esta janela, entra na função "f_FormClosed11()" que chama a função "imprimir_saída()" onde é inserido, na caixa de imagem, o símbolo do constituinte que foi previamente pressionado e a saída escolhida pelo utilizador.

Ainda na Figura 55 pode ainda observar-se que, se for pressionado o botão que está marcado na zona com o número 3, e ao ser pressionada uma caixa de imagem, entra na função "simbolo_imagem()". Esta, chama a janela secundária intitulada como *Form*12, janela "Opções Diversas *Ladder*", onde é escolhida uma opção diversa entre o *Set/ Reset* a uma etapa, um contador, um comparador ou um temporizador. Fechada esta janela, entra na função "f_FormClosed12()" que chama a função "imprimir_caixa()" onde é inserido, na caixa de imagem, o símbolo correspondente à opção escolhida pelo utilizador.

Caso seja pressionado o botão de eliminar, marcado com o número 4, é chamada a janela secundária intitulada como *Form*13, janela "Eliminar *Ladder*", onde são eliminadas partes

constituintes do código *Ladder*. Fechada esta janela, entra na função "f_FormClosed13()" onde são limpas todas as caixas de imagem e novamente preenchidas com as partes não eliminadas pelo utilizador.

Sendo pressionado qualquer outro botão disponível no painel à direita desta janela, é inserida, na caixa de imagem selecionada, o constituinte previamente pressionado sem abrir uma janela secundária.

Na Figura 56 observa-se o menu programar. Caso seja pressionado o botão "Compilar" é chamada a função "compilarToolStripMenuItem_Click()" que, caso o projeto a ser elaborado seja na linguagem *Grafcet*, chama a função "erros_grafcet()", caso seja elaborado na linguagem *Ladder*, chama a função "erros_ladder()" e caso seja na linguagem C, chama a função "compilar()".

Figura 56 - *Form*1, Menu Programar

Caso seja pressionado o botão "Passar para C" é chamada a função, mais importante e trabalhosa deste projeto, "passarParaCToolStripMenuItem_Click()" que, caso o projeto seja elaborado na linguagem *Grafcet* ou *Ladder,* é analisado todo o código e passado para a linguagem C para posterior programação do autómato.

Caso seja pressionado o botão "Programar" é chamada a janela secundária intitulada como *Form*8, janela "Programação", onde é escolhida a porta onde o autómato, a programar, se encontra ligado. Fechada esta janela, entra na função "f_FormClosed8()" que inicia a programação do autómato.

Figura 57 - Form1, Janela código na linguagem C

Na Figura 57 observa-se a janela de código na linguagem C. No painel da esquerda são apresentados os módulos usados no projeto e no painel da direita é apresentado o código desse mesmo módulo. No painel da esquerda não aparece tabela de símbolos, assim sendo as variáveis têm de ser definidas no código como entrada, saída ou mesmo internas.

Como se pode observar, ainda na Figura 57, existe um painel inferior que neste caso contém a informação que a programação foi concluída com êxito.

4.4.1.2. Janela "Novo Ficheiro"

A *Form*2 é a janela de "Novo Ficheiro" que, como referido anteriormente, é chamada na *Form*1, janela "Principal". Nesta janela existe apenas uma função "fechar_novo_ficheiro()" que, ao ser clicado o botão "Criar", esta função guarda nas variáveis globais as form's, o nome do ficheiro, o tipo de autómato e a linguagem de programação do projeto a ser criado. Na Figura 58 pode observar-se a função pertencente a esta *form.*

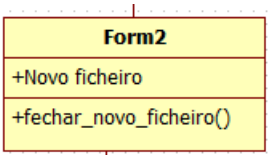


Figura 58 - Função da *Form*2

Após esta janela ser fechada, no botão "Criar", o programa volta a *Form*1 iniciando um novo projeto na linguagem selecionada. Caso seja fechada no botão "fechar janela" o programa volta a *Form*1 não iniciando qualquer tipo de projeto.

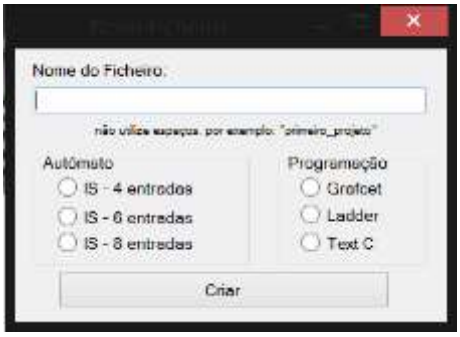


Figura 59 - *Form*2, Novo Ficheiro

Na Figura 59 pode observar-se a Form2, janela "Novo Ficheiro", que contém uma caixa de texto para o nome do ficheiro, os botões para a escolha do tipo de autómato e linguagem de programação e um botão para criar o projeto.

4.4.1.3. Janela "Sobre"

A *Form*3 é a janela do "Sobre". É chamada na *Form*1, janela "Principal", e contém apenas uma função: "fechar_sobre()". Após esta janela ser fechada no botão "Ok", o programa volta à *Form*1 continuando o que estava a ser feito nesta mesma *Form*. Na Figura 60 pode observar-se a função pertencente a esta *form*.

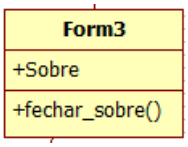


Figura 60 - Função da *Form*3

Na Figura 61 pode observar-se a *Form*3, janela "Sobre", contendo a informação do ambiente de desenvolvimento multilinguagem, bem como um botão para fechar a *Form*.

Figura 61 - *Form*3, Sobre

4.4.1.4. Janela "Símbolos *Grafcet*"

A *Form*4 é a janela "Adicionar *Grafcet*". É chamada na *Form*1, janela "Principal", para inserir símbolos na tabela na linguagem *Grafcet*. Na Figura 62 podem observar-se as funções pertencentes a esta *form*.

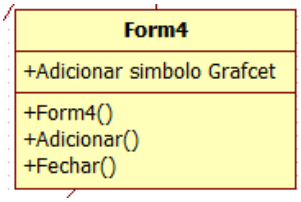


Figura 62 - Funções da *Form*4

Na função "Form4()" são limpas as variáveis globais utilizadas nesta *Form*. São igualmente inseridas, na lista de posições, as entradas/saídas disponíveis no autómato, dependendo do tipo do mesmo, seja de 4, 6 ou 8 entradas/saídas.

A função "Adicionar()" é chamada quando é pressionado o respetivo botão, "Adicionar". Chama a função "Fechar()" que guarda nas variáveis globais, o nome do símbolo, o tipo do símbolo (entrada, interno, saída ou temporizador), a posição de ligação no autómato, se for uma entrada ou saída, o seu valor inicial e possíveis comentários. No fim desta função é dada a ordem de fecho desta janela.

Após esta janela ser fechada, o programa volta à *Form*1 onde insere na tabela de símbolos do código Grafcet as variáveis globais que foram preenchidas na *Form*4.

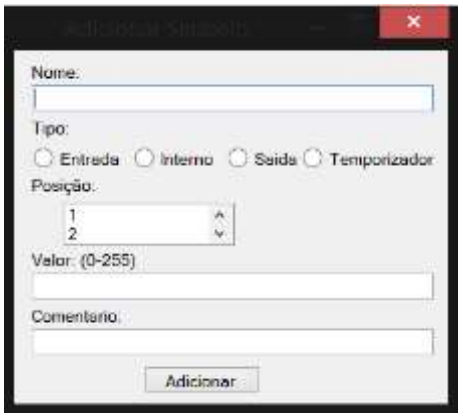


Figura 63 - *Form*4, Adicionar Símbolo

Na Figura 63 pode observar-se a *Form*4, janela "Adicionar Símbolo Grafcet", que contém uma caixa de texto para o nome do símbolo, botões para selecionar o tipo do símbolo, a posição de ligação no autómato a ser programado, uma caixa de texto para o seu valor inicial e uma caixa de texto para um possível comentário.

4.4.1.5. Janela "Transição *Grafcet*"

A *Form*5 é a janela "Transição" na linguagem *Grafcet* e é chamada na *Form*1, janela "Principal". Na Figura 64 podem observar-se as funções pertencentes a esta *form*.

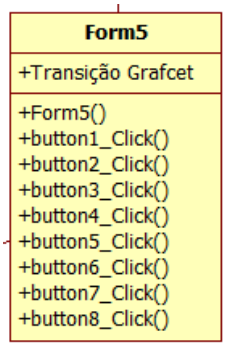


Figura 64 - Funções da *Form*5

Na função "Form5()" são inseridas todas as variáveis disponíveis no projeto na lista à esquerda desta janela. As funções "button2_Click()", "button4_Click()", "button5_Click()", "button6_Click()", "button7_Click()" e "button8_Click()" inserem, na caixa de texto o símbolo "*" correspondente à instrução lógica "e", o símbolo "+" correspondente à instrução lógica "ou", o símbolo "(" que significa "abertura de parenteses", o símbolo ")" que significa "fecho de parenteses", o símbolo "=" que significa "igual" e o símbolo "!=" que significa "diferente", respetivamente.

A função "button1_Click()" adiciona, à caixa de texto, a variável que estiver selecionada na lista e o botão que estiver selecionado: "Ligado", "Desligado", "Igual a", "Diferente de".

A função "button3_Click()" guarda o que estiver na caixa de texto na variável global "transição_grafcet" e dá ordem de fecho da janela.

Fechada a janela, o programa volta à *Form*1 onde insere, no código *Grafcet*, a imagem de transição e a respetiva transição preenchida na *Form*5.

Figura 65 - *Form*5, Transição

Na Figura 65 pode observar-se a *Form*5, janela "Transição *Grafcet*", que contém uma lista com as variáveis disponíveis no projeto, botões para construção automática da transição ("Ligado", "Desligado", "Igual a", "Diferente de"), um botão que adiciona à caixa de texto a construção

automática, botões para construção manual da transição ("*", "+", "(" e ")"), uma caixa de texto onde é apresentada a transição e um botão para finalizar.

4.4.1.6. Janela "Etapa *Grafcet*"

A *Form*6 é a janela "Etapa" na linguagem *Grafcet* e é chamada na *Form*1, janela "Principal". Na Figura 66 podem observar-se as funções pertencentes a esta *form*.

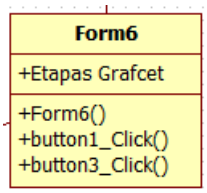


Figura 66 - Funções da *Form*6

Na função "Form6()" são inseridas todas as variáveis disponíveis no projeto na lista à esquerda desta janela. Na função "button1_Click()" é adicionado na tabela à direita, e na variável global "etapa_grafcet", a variável que estiver selecionada na lista juntamente com a opção escolhida através dos botões "Ligado", "Incrementar", "Decrementar" e "Inicializar". Na função "button3_Click()" é fechada a respetiva janela. Após esta janela ser fechada, o programa volta a *Form*1 onde insere no código *Grafcet,* a imagem de etapa e as respetivas ativações nesta etapa preenchida na *Form*6.

Figura 67 - *Form*6, Etapas

Na Figura 67 pode observar-se a *Form*6, janela "Etapa *Grafcet*", que contém uma lista com a variáveis disponíveis no projeto, botões para ligar, incrementar, decrementar ou inicializar a variável selecionada e uma tabela onde são inseridos conjuntos de opções selecionadas.

4.4.1.7. Janela "Eliminar *Grafcet*"

A *Form*7 é a "Eliminar" na linguagem *Grafcet*, e é chamada na *Form*1, janela "Principal". Na Figura 68 podem observar-se as funções pertencentes a esta *form*.

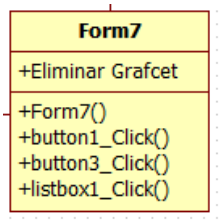


Figura 68 - Funções da *Form7*

Na função "Form7()" são inseridos todos os constituintes do código na linguagem *Grafcet*. A função "button1_Click()" é chamada quando pressionado o botão "Adicionar" e adiciona o elemento selecionado da lista da esquerda à lista da direita removendo-o da primeira lista. A função "listbox1_Click()" apresenta, nas duas caixas de imagem disponíveis nesta janela, a parte do código selecionado na lista da esquerda.

A função "button3_Click()" é chamada quando pressionado o botão "Eliminar" e guarda, num ficheiro de texto, o código sem os constituintes que pretendem ser eliminadas pelo utilizador, concluindo com o fecho desta janela.

Após esta janela ser fechada, o programa volta a *Form*1, é eliminado todo o código *Grafcet* e são inseridos apenas os constituintes que ficaram no ficheiro criado na *Form*6.

Figura 69 - *Form*7, Eliminar

Na Figura 69 pode observar-se a *Form*7, janela "Eliminar *Grafcet*", que contém duas listas: a lista da esquerda que contém todos os constituintes do código na linguagem *Grafcet* e a lista da direita que contém os constituintes que o utilizador pretende eliminar. Contém também duas caixas de imagem que apresentam o constituinte selecionado na lista da esquerda.

4.4.1.8. Janela "Programação"

A *Form*8 é a janela de "Programação" e é chamada na *Form*1, janela "Principal". Na Figura 70 podem observar-se as funções pertencentes a esta *form*.

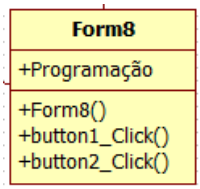


Figura 70 - Funções da *Form*8

Na função "Form8()" são inseridas, na caixa de seleção, as portas disponíveis e na função "button1_Click", que é chamada quando pressionado o botão de "*Re-scan*", são limpas todas as portas e novamente inseridas as portas disponíveis.

A função "button2_Click()" é chamada quando pressionado o botão "Ok" e guarda, na variável global "porta_com", a porta selecionada, concluindo com o fecho desta janela.

Após esta janela ser fechada, o programa volta à *Form*1 onde é programado o autómato que se encontra ligado à porta USB selecionada na *Form*8.

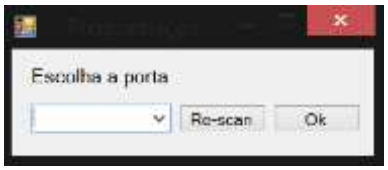


Figura 71 - *Form*8, Programação

Na Figura 71 pode observar-se a *Form*8, janela "Programação", que contém uma caixa com as portas disponíveis, um botão para reencontrar as portas disponíveis e um botão para selecionar a porta escolhida.

4.4.1.9. Janela "Símbolos *Ladder*"

A *Form*9 é a janela de símbolos na linguagem *Ladder*, e é chamada na *Form*1, janela "Principal". Na Figura 72 podem observar-se as funções pertencentes a esta *form*.

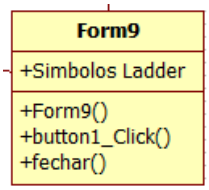


Figura 72 - Funções da *Form*9

Na função "Form9()" são limpas as variáveis globais utilizadas nesta *Form* e inseridas, na lista de posições, as entradas ou saídas disponíveis no autómato dependendo do tipo do mesmo, seja de 4, 6 ou 8 entradas/saídas.

A função "button1_Click()" é chamada quando pressionado o botão "Adicionar" e chama a função "fechar()". Na função "fechar()" são guardadas as variáveis globais, o nome do símbolo, o tipo do símbolo (entrada, interno, saída, temporizador ou contador), a posição de ligação no autómato se for uma entrada ou saída, o seu valor inicial e possíveis comentários. No fim desta função é dada a ordem de fecho desta janela.

Figura 73 - *Form9*, Símbolos

Após esta janela ser fechada, o programa volta à *Form*1 onde insere, na tabela de símbolos do código *Ladder*, as variáveis globais que foram preenchidas na *Form9*.

Na Figura 73 pode observar-se a *Form9*, janela "Símbolos *Ladder*", que contém uma caixa de texto para o nome do símbolo, botões para selecionar o tipo do símbolo, a posição de ligação no autómato a ser programado, uma caixa de texto para o seu valor inicial e uma caixa de texto para um possível comentário.

4.4.1.10. Janela "Entrada *Ladder*"

A *Form*10 é a janela de "Entrada" na linguagem *Ladder*, e é chamada na *Form*1, janela "Principal". Na Figura 74 podem observar-se as funções pertencentes a esta *form*.

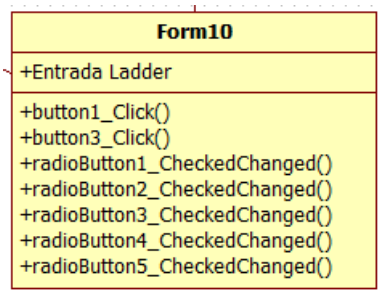


Figura 74 - Funções da *Form*10

As funções “radioButton1_CheckedChanged()” a “radioButton5_CheckedChanged()” quando pressionados os respetivos botões a elas associados apresentam, na lista desta *Form,* as variáveis do tipo Entrada, Interno, Contador, Saída e Temporizador, respetivamente.

A função “button1_Click()” é chamada quando pressionado o botão “Adicionar” e adiciona à caixa de texto a variável que estiver selecionada na lista e a opção “ligado” ou “igual a”.

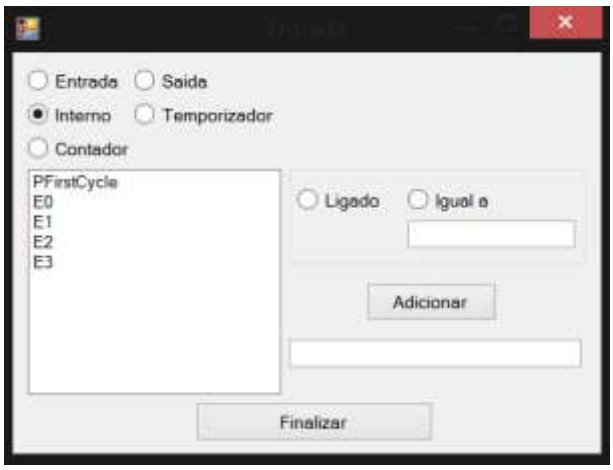


Figura 75 - *Form*10, Entrada

A função “button3_Click()” guarda, na variável global “entradas_ladder”, o conteúdo da caixa de texto e dá ordem de fecho desta janela. Após esta janela ser fechada, o programa volta a *Form*1 onde insere no código *Ladder*, a imagem de uma entrada e a respetiva variável escolhida na *Form*10.

Na Figura 75 pode observar-se a *Form*10, janela “Entrada *Ladder*”, que contém cinco botões para selecionar o tipo de variáveis, dois botões para escolher entre “ligado” ou “igual a um valor exato”, um botão para adicionar a opção escolhida, uma caixa de texto que apresenta a solução escolhida e um botão para fechar a janela.

4.4.1.11. Janela “Saída *Ladder*”

A *Form*11 é a janela de “Saídas” na linguagem *Ladder*, e é chamada na *Form*1, janela “Principal”. Na Figura 76 podem observar-se as funções pertencentes a esta *form*.

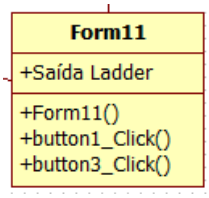


Figura 76 - Funções da *Form*11

Na função "Form11()" são inseridas todas as variáveis do tipo interno ou saída, disponíveis no projeto, na lista à esquerda nesta janela. Na função "button1_Click()" é inserida, na caixa de texto, a variável que estiver selecionada na lista. A função "button3_Click()" guarda, na variável global "saídas_ladder", o conteúdo da caixa de texto e dá ordem de fecho desta janela. Após esta janela ser fechada, o programa volta a *Form*1 onde insere, no código *Ladder,* a imagem de uma saída e a respetiva variável escolhida na *Form*11.

Figura 77 - *Form*11, Saída

Na Figura 77 pode observar-se a *Form*11, janela "Saída *Ladder*", que contém uma lista com as variáveis que podem ser ativadas, variáveis internas e saídas, um botão para adicionar a opção escolhida, uma caixa de texto que apresenta a solução escolhida e um botão para fechar a janela.

4.4.1.12. Janela "Opções Diversas *Ladder*"

A *Form*12 é a janela de "Opções Diversas" na linguagem *Ladder*, e é chamada na *Form*1, janela "Principal". Na Figura 78 podem observar-se as funções pertencentes a esta *form*.

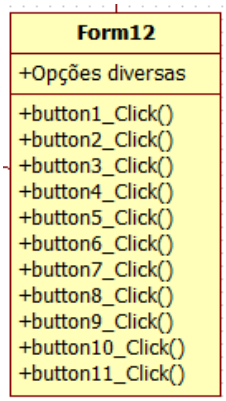


Figura 78 - Funções da *Form*12

As funções "button1_Click()" a "button5_Click()" inserem, na lista à esquerda, todas as variáveis que podem ser usadas como: *Set*, *Reset*, Contador, Comparador e Temporizador, respetivamente.

As funções “button7_Click()” a “button11_Click()” inserem na caixa de texto a variável que estiver selecionada na lista, dependendo se tiver sido selecionado anteriormente o *Set*, *Reset*, Contador, Comparador ou Temporizador, respetivamente, guardando também, na variável global “caixa_ladder”, a opção escolhida e a variável escolhida.

A função “button6_Click()” dá ordem de fecho desta janela. Após o fecho, o programa volta à *Form*1 onde insere, no código *Ladder,* a imagem correspondente à opção escolhida na *Form*12.

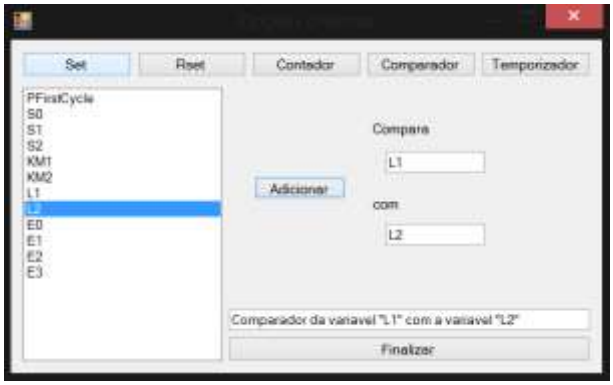


Figura 79 - *Form*12, Opções diversas

Na Figura 79 pode observar-se a *Form*12, janela “Opções Diversas *Ladder*”, onde contém uma lista com as variáveis que podem ser usadas pelo comparador, cinco botões para selecionar a opção desejada entre *Set*, *Reset*, Contador, Comparador e Temporizador, um botão para adicionar a variável selecionada na lista, e um botão para fechar a janela.

4.4.1.13. Janela “Eliminar *Ladder*”

A *Form*13 é a “Eliminar” na linguagem *Ladder*, e é chamada na *Form*1, janela “Principal”. Na Figura 80 podem observar-se as funções pertencentes a esta *form*.

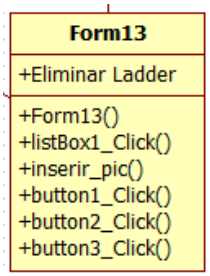


Figura 80 - Funções da *Form*13

Na função “Form13()” são inseridos todos os constituintes do código na linguagem *Ladder* na lista à esquerda desta *Form*. A função “listbox1_Click()” chama a função “inserir_pic()” para cada uma das nove caixas de imagem disponíveis nesta janela. A função “inserir_pic()” insere o

constituinte do código na linguagem *Ladder* na caixa de imagem pedida na função "listbox1_Click()".

A função "button1_Click()" é chamada quando pressionado o botão ">>" e adiciona o elemento selecionado da lista da esquerda à lista da direita removendo-o da primeira. A função "button2_Click()" é chamada quando pressionado o botão "<<" e adiciona o elemento selecionado da lista da direita à lista da esquerda removendo-o da lista da direita. A função "button3_Click()" é chamada quando pressionado o botão "Eliminar" e guarda, num ficheiro de texto, o código sem os constituintes que pretendem ser eliminadas pelo utilizador, concluindo com o fecho desta janela. Após esta janela ser fechada, o programa volta à *Form*1, é eliminado todo o código *Ladder* e são inseridos apenas os constituintes que ficaram no ficheiro criado na *Form*13.

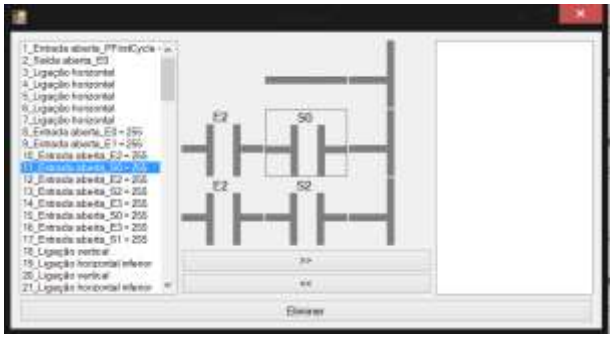


Figura 81 - *Form*13, Eliminar

Na Figura 81 pode observar-se a *Form*13, janela "Eliminar *Ladder*", que contém duas listas: a lista da esquerda que contém todos os constituintes do código na linguagem *Ladder* e a lista da direita que contém os constituintes que o utilizador pretende eliminar. Contém também nove caixas de imagem que apresentam o constituinte selecionado na lista da esquerda bem como o seu envolvente.

Capítulo 5

5. Resultados Obtidos e Discussão

Neste capítulo serão apresentados os resultados obtidos nos testes realizados bem como a discussão dos mesmos. Para uma melhor depuração dos algoritmos desenvolvidos recorreu-se a realização de apenas seis exercícios, devido a vasta extensão da explicação dos exercícios pertencentes ao livro de exercícios da cadeira de Automação, lecionada no terceiro ano do Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores, disponível no anexo A. Ambos os exemplos/exercícios apresentados neste capítulo foram elaborados na linguagem *Grafcet* e *Ladder* sendo, neste último, resolvidos através do método de equações lógicas e através do método de funções *Set/ Reset*.

Nos apêndices são apresentados os códigos em linguagem C de cada exercício, gerados pelo ambiente de desenvolvimento multilinguagem, por forma a ser possível programar o respetivo autómato programável. Os exercícios recorrem ao uso de duas bibliotecas, uma com funções de comunicação pela porta série, "Funções_USART.c" no Apêndice A e "Funções_USART.h" no Apêndice B, outra com funções para inicialização e utilização dos canais ADC, "Funções_ADC.c" no Apêndice C e "Funções_ADC.h" no Apêndice D.

5.1 Inversão de Marcha de Motor Trifásico

Neste primeiro exercício pretende-se um automatismo que permita a inversão de marcha de um motor trifásico com recurso a três botões, dois relês e dois sinalizadores luminosos.

Como entradas no sistema, os botões funcionam da seguinte forma: sendo pressionado o botão "s1" efetua-se a marcha à direita; sendo pressionado o botão "s2" efetua-se a marcha à esquerda e sendo pressionado o botão "s0" efetua-se a paragem do motor.

Como saídas no sistema, os sinalizadores luminosos indicam a marcha a ser executada pelo motor: um sinalizador para marcha à direita e outro sinalizador para marcha à esquerda. Ainda como saídas no sistema são usados dois relês, devido à inversão do sentido de rotação dos motores assíncronos trifásicos ser realizada através da troca de fases que alimentam o motor, o que se pode observar na Figura 82. [Anexo A]

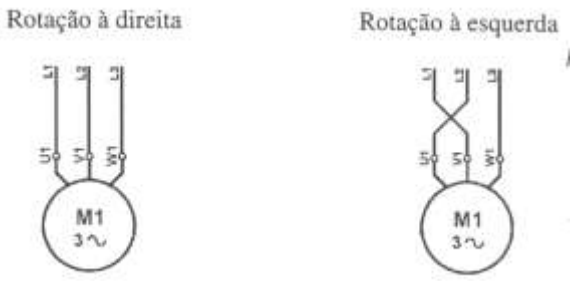


Figura 82 - Esquema de inversão de rotação de um motor trifásico [Anexo A]

5.1.1. *Grafcet*

Recorrendo ao ambiente de desenvolvimento, foi elaborado o projeto na linguagem de *Grafcet* do respetivo enunciado. Na Figura 83 observa-se a tabela de símbolos que descreve o nome das entradas/saídas utilizadas neste projeto, bem como as posições em que estas serão ligadas no autómato.

Nomes	Tipos	Posições	Comentarios
S1	Entrada	1	marcha direita
S2	Entrada	2	marcha esquerda
S0	Entrada	3	paragem
KM1	Saída	1	rodar a direita
KM2	Saída	2	roda a esquerda
L1	Saída	3	lâmpada roda a direita
L2	Saída	4	lâmpada roda a esquerda

Figura 83 - Tabela de símbolos *Grafcet* do primeiro exercício

Estando, então, a tabela de símbolos bem definida é necessário elaborar o diagrama em *Grafcet* que permita o funcionamento do automatismo como pretendido no enunciado.

Na Figura 84 observa-se o código em linguagem *Grafcet* constituído por 3 etapas fundamentais. Na primeira etapa, “etapa 1”, o motor está parado e as lâmpadas sinalizadores desligadas. Caso o botão “s1”- marcha à direita, seja pressionado e o botão “s0”- paragem, não esteja pressionado o automatismo passa para a segunda etapa, ligando o relê que permite a marcha à direita e a respetiva lâmpada sinalizadora. O automatismo permanece na “etapa 2” até que seja pressionado o botão “s0”- paragem, ou o botão “s2”- marcha à esquerda, passando a estar ativa a “etapa 1”.

Caso o botão "s2"- marcha à direita, seja pressionado e o botão "s0"- paragem, não seja pressionado, o automatismo passa para a terceira etapa, ligando o relê que permite a marcha à esquerda e a respetiva lâmpada sinalizadora. A "etapa 3" mantem-se ativa até que seja pressionado o botão "s0"- paragem, ou o botão "s1"- marcha à esquerda, passando a estar ativa a "etapa 1".

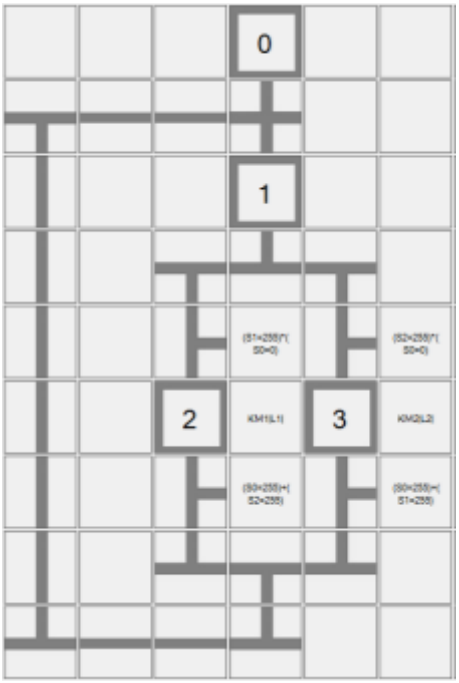


Figura 84 - Código em linguagem *Grafcet* do primeiro exercício

Por fim o código em *Grafcet* é passado para C por forma a ser possível programar o respetivo autómato programável, podendo este ser encontrado no Apêndice E.

5.1.2. *Ladder*

No ambiente de desenvolvimento foi elaborado o projeto na linguagem de *Ladder* do respetivo enunciado. Na Figura 85 observa-se a tabela de símbolos que descreve o nome das entradas/saídas/variáveis internas utilizadas neste projeto, bem como as posições em que estas serão ligadas no respetivo autómato. Num projeto em *Ladder* é sempre necessário definir como variáveis internas as etapas do projeto a ser desenvolvido.

Nome	Tipo	Posição	Comentário
PFirstCycle	Interno	0	Primeiro Ciclo
S0	Entrada	1	paragem
S1	Entrada	2	marcha direita
S2	Entrada	3	marcha esquerda
KM1	Saída	1	rodar a direita
KM2	Saída	2	rodar a esquerda
L1	Saída	3	lâmpada roda a direita
L2	Saída	4	lâmpada roda a esquerda
E0	Interno	1	etapa 0
E1	Interno	1	etapa 1
E2	Interno	1	etapa 2
E3	Interno	1	etapa 3

Figura 85 - Tabela de símbolos *Ladder* do primeiro exercício

Estando a tabela de símbolos bem definida é necessário, então, elaborar o diagrama em *Ladder* que permitirá o funcionamento do automatismo, como pretendido no enunciado. No primeiro ponto proceder-se-á à elaboração através do método de equações lógicas e no segundo ponto através do método de funções *Set/ Reset*.

5.1.2.1. Método de Equações Lógicas

No método de equações lógicas, é necessário definir as equações para o correto funcionamento do automatismo. De referir que as equações lógicas englobam a ativação/desativação das etapas e saídas de todo o sistema.

A primeira equação lógica tem como entrada o primeiro ciclo de execução do projeto e ativa a "etapa 0", sendo dada pela Equação (1). A segunda equação lógica tem como entrada todas as possibilidades da "etapa 1" ficar ativa, sendo dada pela Equação (2).

$$E_0 = P\_First\_Cycle \quad (1)$$

$$E_1 = (E_1 + E_0 + E_2 * S0 + E_2 * S2 + E_3 * S0 + E_3 * S1) * \overline{E_2} * \overline{E_3} \quad (2)$$

Na Figura 86 observa-se o código do diagrama *Ladder* correspondente às duas primeiras equações: Equação (1) e Equação (2).

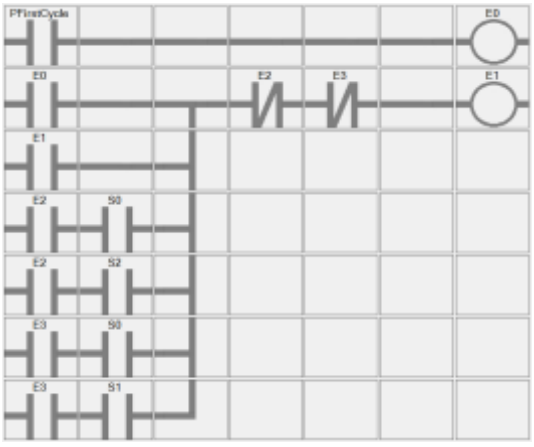


Figura 86 - Primeira parte das equações lógicas do primeiro exercício

A terceira equação lógica tem como entrada todas as possibilidades da "etapa 2" ficar ativa, sendo dada pela Equação (3). A quarta equação lógica e última, referente a ativação das etapas, tem como entrada todas as possibilidades da "etapa 3" ficar ativa, sendo dada pela Equação (4). A ativação das saídas, equações lógicas quinta e sexta, são dadas pela Equação (5) e Equação (6) respetivamente.

$$E_2 = (E_2 + E_1 * S1 * \overline{S0}) * \overline{E_3} \quad (3)$$

$$E_3 = (E_3 + E_1 * S2 * \overline{S0}) * \overline{E_2} \quad (4)$$

$$KM1 + L1 = E_2 \quad (5)$$

$$KM2 + L2 = E_3 \quad (6)$$

Na Figura 87 observa-se o código do diagrama *Ladder* correspondente às quatro equações lógicas: Equação (3), Equação (4), Equação (5) e Equação (6).

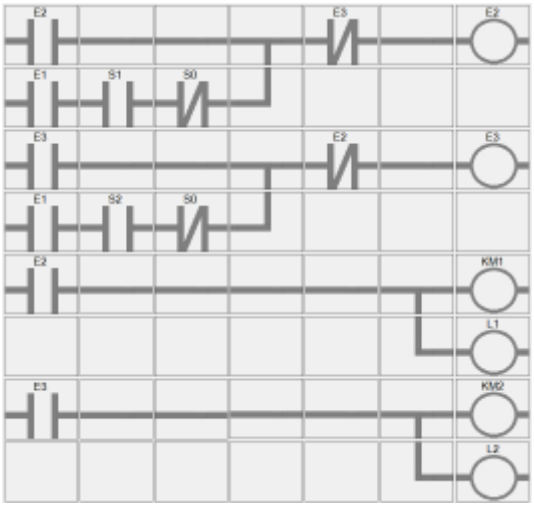


Figura 87 – Segunda parte das equações lógicas do primeiro exercício

As seguintes equações: sétima, oitava, nona e decima, representam a descativação de todas as etapas do projeto, correspondendo às equações: Equação (7), Equação (8), Equação (9) e Equação (10).

$$\overline{E_0} = E_1 \quad (7)$$

$$\overline{E_1} = E_2 + E_3 \quad (8)$$

$$\overline{E_2} = E_1 + E_3 \quad (9)$$

$$\overline{E_3} = E_1 + E_2 \quad (10)$$

Na Figura 88 observa-se o código do diagrama *Ladder* correspondente a estas quatro últimas equações lógicas.

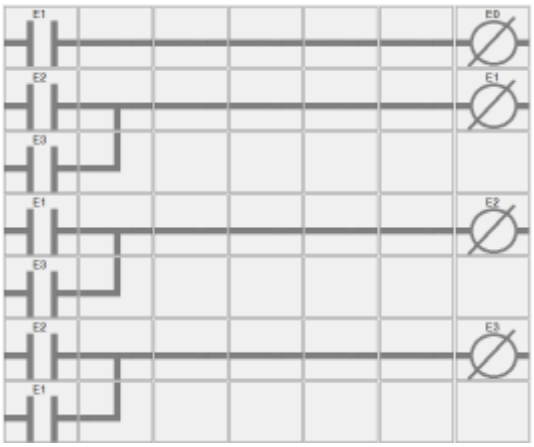


Figura 88 – Terceira parte das equações lógicas do primeiro exercício

Por fim o código em *Ladder* é passado para C por forma a ser possível programar o respetivo autómato programável, podendo este ser encontrado no Apêndice F.

5.1.2.2. Método de Funções *Set/ Reset*

O método de funções *Set/ Reset* partindo do diagrama de *Grafcet*, é de fácil implementação. As mesmas transições usadas no Grafcet são as equações que ativam a etapa seguinte, *Set*, e desativam a etapa anterior, *Reset*. Comparando este método ao método de equações lógicas, o código *Ladder* é mais facilmente percetível e menos extenso.

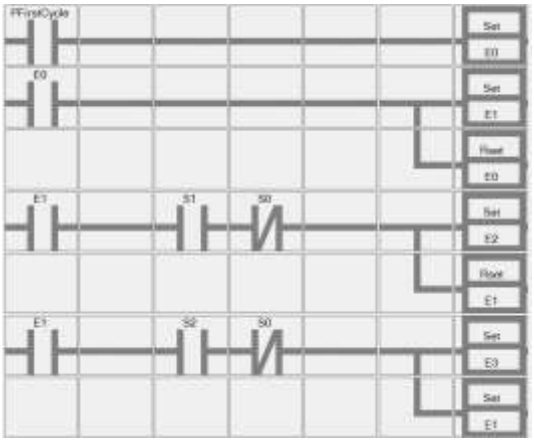


Figura 89 - Primeira parte das funções *Set/ Reset* do primeiro exercício

Na Figura 89 observa-se a primeira parte do método das funções de *Set/ Reset*. Os dois primeiros degraus do projeto apenas são executados no primeiro ciclo do projeto ativando a "etapa 0" e posteriormente a "etapa 1". O terceiro e quarto degrau tem ambos a "etapa 1" ativa como condição mas, no terceiro degrau, caso o botão "s1" seja pressionado e o botão "s0" não seja pressionado fica ativa a "etapa 2" e, no quarto degrau, caso o botão "s2" seja pressionado e o

botão "s0" não seja pressionado fica ativa a "etapa 2". Em ambos os casos a "etapa 1" deixa de estar ativa.

Na Figura 90 observa-se a segunda e última parte do método das funções de *Set/ Reset*. O quinto degrau depende da "etapa 2" estar ativa e do botão "s0" ou "s2" estar pressionado. Caso isto aconteça é ativada a "etapa 1" e desativada a "etapa 2". O sexto degrau é semelhante ao quinto mas dependendo da "etapa 3" ativa, e não da "etapa 2", ativando na mesma a "etapa 1" mas desativando a "etapa 3".

Os últimos dois degraus referem-se às ativações das saídas nas respetivas etapas. Estes dois degraus são iguais tanto no método de equações lógicas como no método de funções *Set/ Reset* pois, independentemente do método usado, as saídas são ativadas da mesma forma.

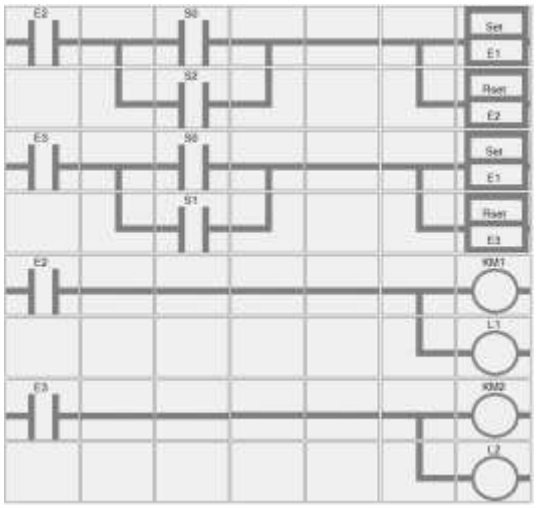


Figura 90 - Segunda parte das funções *Set/ Reset* do primeiro exercício

Para ser possível programar o respetivo autómato programável, o código em *Ladder* é passado para C, podendo ser encontrado no Apêndice G.

5.2 Portão Automático

Neste segundo exercício é pretendido um automatismo para comandar à distância a abertura e fecho de um portão automático (Figura 91) com recurso a dois motores, "M1" e "M2".

Através de um comando à distância é fornecida a ordem para abertura do portão, recebida pelo recetor "Rx", iniciando a abertura do portão simultaneamente com uma lâmpada sinalizadora. Nos êmbolos roscados dos motores são usados interruptores de fim de curso, "fc1" e "fc2" que, quando pressionados, os motores param e a lâmpada é desligada.

O portão permanece aberto durante 60s iniciando automaticamente o seu fecho, ligando novamente a lâmpada sinalizadora. O fecho do portão, como não existem sensores de fim de curso, é realizado durante 20s. Durante o fecho automático do portão, caso o exista uma ordem fornecida pelo comando para abertura "Rx", ou a barreira de infravermelhos "IR" seja interrompida, o portão volta a abrir. [Anexo A]

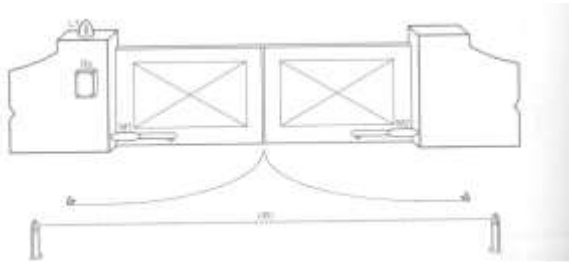


Figura 91 - Portão Automático [Anexo A]

5.2.1. *Grafcet*

Com recurso ao ambiente de desenvolvimento, foi elaborado o projeto na linguagem de *Grafcet* do respetivo enunciado. Na Figura 92 observa-se a tabela de símbolos que descreve o nome das entradas/saídas utilizadas neste projeto, bem como as posições em que estas serão ligadas no respetivo autómato. Este projeto pressupõe também a existência de um temporizador cuja posição não é relevante.

Nomes	Tipos	Posições	Comentarios
Rx	Entrada	1	sinal de inicio de abertura
Fc1	Entrada	2	fim de curso motor 1
Fc2	Entrada	3	fim de curso motor 2
Ir	Entrada	4	barreira de infravermelhos
Li	Saida	1	lampada sinalizadora
M1	Saida	2	motor 1
M2	Saida	3	motor 2
Temp	Temporizador	1	temporizador

Figura 92 - Tabela de símbolos *Grafcet* do segundo exercício

A elaboração do diagrama em *Grafcet* é possível após a definição da tabela de símbolos, permitindo o funcionamento do automatismo como pretendido no enunciado.

Na Figura 93 observa-se o código em linguagem *Grafcet* constituído por 4 etapas fundamentais. A primeira etapa, "etapa 1", apenas é ativada caso uma das seguintes transições: "Rx", "Rx" ou "Ir" e "Temp=40" sejam validadas. Neste caso são ativos os dois motores para abertura e ligada a lâmpada sinalizadora.

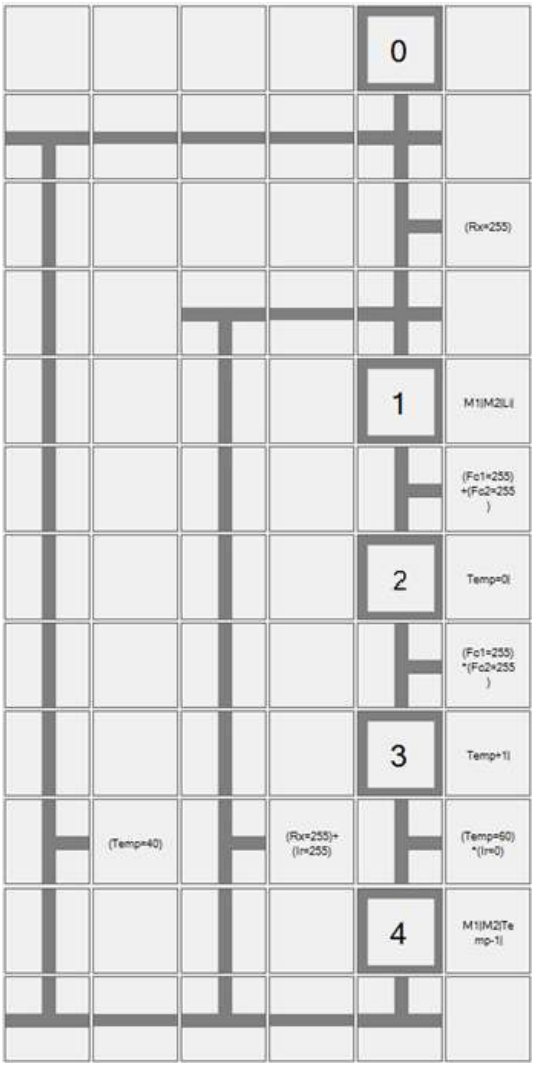


Figura 93 - Código em linguagem *Grafcet* do segundo exercício

A "etapa 2" é ativada quando um dos sensores fim de curso "fc1" ou "fc2" for pressionado. Neste caso, o temporizador, é inicializado a zero.

A "etapa 3" é ativada quando ambos os sensores fim de curso "fc1" e "fc2" forem pressionados, sendo iniciadas as contagens incrementais do temporizador.

A última etapa, "etapa 4", é ativada quando o temporizador chega aos 60s sem que a barreira de infravermelhos seja interrompida. Neste caso é iniciado o fecho do portão bem como as contagens decrementais do temporizador.

Nesta quarta etapa existem duas opções no decorrer do fecho do portão: caso o temporizador chegue ao fim, os motores param ficando o portão fechado; caso seja pressionado o comando à distância antes do temporizador terminar a contagem ou interrompida barreira de infravermelhos, o portão volta abrir.

Concluído o projeto, o código em *Grafcet* é passado para C por forma a ser possível programar o respetivo autómato programável, podendo este ser encontrado no Apêndice H.

5.2.2. *Ladder*

Criando um novo projeto no ambiente de desenvolvimento, na linguagem de *Ladder*, é possível elaborar o código do respetivo enunciado. Na Figura 94 observa-se a tabela de símbolos que descreve o nome das entradas/saídas/variáveis internas e temporizadores utilizados neste projeto, bem como as posições em que estas serão ligadas no respetivo autómato. Como já foi referido, num projeto em *Ladder* é sempre necessário definir como variáveis internas as etapas do projeto a ser desenvolvido.

Nome	Tipo	Posição	Comentário
P_First_Cycle	Interno	0	Primeiro Ciclo
Rx	Entrada	1	sinal de início de abertura
Fc1	Entrada	2	fim de curso motor 1
Fc2	Entrada	3	fim de curso motor 2
Ir	Entrada	4	barreira de infravermelhos
Li	Saída	1	lampada sinalizadora
M1	Saída	2	motor 1
M2	Saída	3	motor 2
Temp	Temporizador	1	temporizador
E0	Interno	1	etapa 0
E1	Interno	1	etapa 1
E2	Interno	1	etapa 2
E3	Interno	1	etapa 3
E4	Interno	1	etapa 4
Temp2	Temporizador	1	temporizador 2

Figura 94 - Tabela de símbolos *Ladder* do segundo exercício

Estando a tabela de símbolos definida é necessário elaborar o diagrama em *Ladder* que permitirá o funcionamento do automatismo, como pretendido no enunciado. No primeiro ponto proceder-se-á à elaboração através do método de equações lógicas e no segundo através do método de funções *Set/ Reset*.

5.2.2.1. Método de Equações Lógicas

No método de equações lógicas é necessário definir as equações para o correto funcionamento do automatismo. Estas equações lógicas englobam a ativação/desativação das etapas e saídas de todo o sistema.

A primeira equação lógica tem como entrada o primeiro ciclo de execução do automatismo, ativando a "etapa 0" e sendo dada pela Equação (11). A segunda equação lógica tem como entrada todas as possibilidades de a "etapa 1" ficar ativa, sendo dada pela Equação (12).

$$E_0 = P\_First\_Cycle \quad (11)$$

$$E_1 = (E_1 + E_0 * Rx + E_4 * Rx + E_4 * Ir + E_4 * Temp2 = 0 * Rx) * \overline{E_2} \quad (12)$$

Na Figura 95 observa-se o código do diagrama *Ladder* correspondente às duas primeiras equações: Equação (11) e Equação (12).

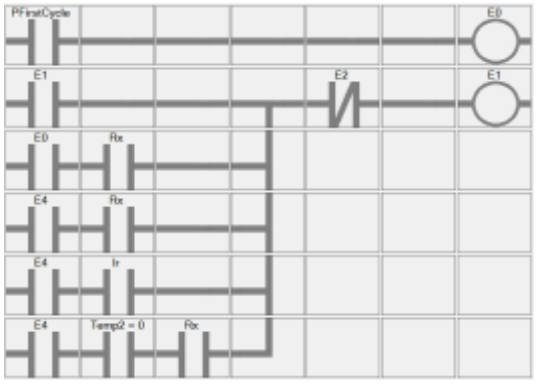


Figura 95 - Primeira parte das equações lógicas do segundo exercício

A terceira equação lógica tem como entrada todas as possibilidades da "etapa 2" ficar ativa, sendo dada pela Equação (13). O mesmo acontece na quarta e quinta, equações lógicas que, tendo como entradas todas possibilidades da "etapa 3" e "etapa 4" passarem a estar ativas, respetivamente. Estas duas equações lógicas são dadas pela Equação (14) e Equação (15) respetivamente.

$$E_2 = (E_2 + E_1 * fc1 + E_1 * fc2) * \overline{E_3} \quad (13)$$

$$E_3 = (E_3 + E_2 * fc1 * fc2) * \overline{E_4} \quad (14)$$

$$E_4 = (E_4 + E_3 * Temp = 0 * Ir) * \overline{E_1} \quad (15)$$

Na Figura 96 observa-se o código correspondente da Equação (13) à Equação (15).

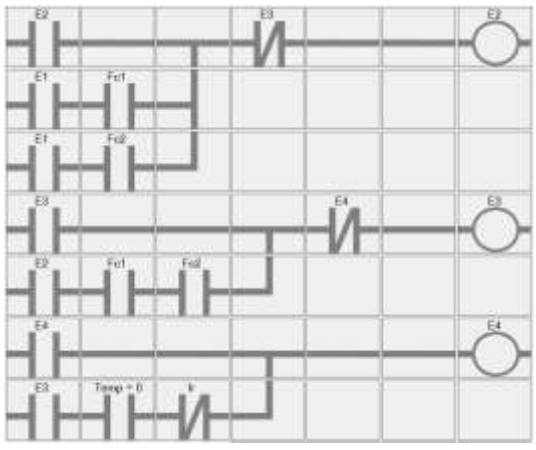


Figura 96 – Segunda parte das equações lógicas do segundo exercício

A ativação das saídas acontece na sexta, sétima e oitava, equações lógicas sendo dadas pelas Equação (16), Equação (17) e Equação (18) respetivamente. Cada equação lógica tem as respetivas saídas a serem ativadas na correspondente etapa.

$$M1 + M2 + Li = E_1 \quad (16)$$

$$Temp - 1\ (\#60) = E_3 \quad (17)$$

$$M1 + M2 + Temp2 - 1\ (\#20) = E_4 \quad (18)$$

Na Figura 97 observa-se o código correspondente as equações: Equação (16), Equação (17) e Equação (18).

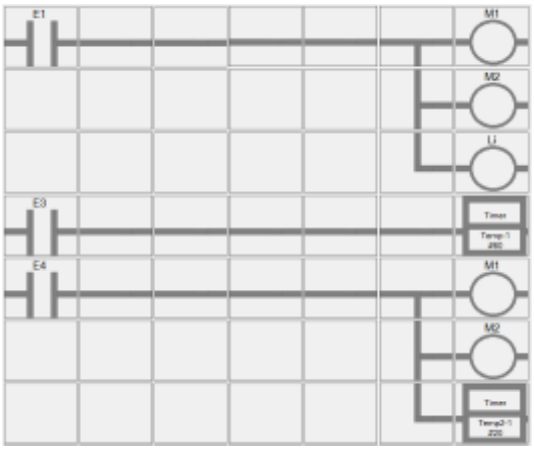


Figura 97 – Terceira parte das equações lógicas do segundo exercício

As seguintes equações: nona, décima, decima-primeira, decima-segunda e decima-terceira, representam a descativação de todas as etapas pertencentes ao projeto, correspondendo às: Equação (19), Equação (20), Equação (21), Equação (22) e Equação (23) respetivamente.

$$\overline{E_0} = E_1 \quad (19)$$

$$\overline{E_1} = E_2 \quad (20)$$

$$\overline{E_2} = E_3 \quad (21)$$

$$\overline{E_3} = E_4 \quad (22)$$

$$\overline{E_4} = E_1 \quad (23)$$

Na Figura 98 observam-se estas últimas cinco últimas equações lógicas.

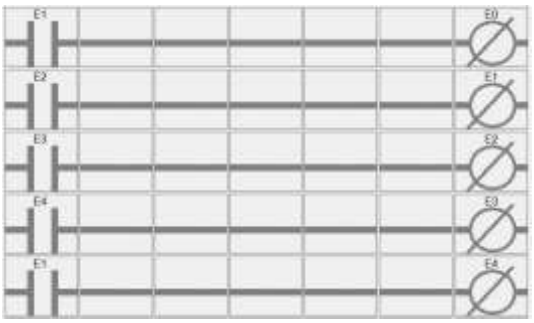


Figura 98 – Quarta parte das equações lógicas do primeiro exercício

Por fim, o código em *Ladder* é passado para C por forma a ser possível programar o respetivo autómato programável, podendo este ser encontrado no Apêndice I.

5.2.2.2. Método de Funções *Set/ Reset*

Este método, partindo do diagrama de *Grafcet*, é de fácil implementação. As transições usadas no *Grafcet* são as equações que ativam a etapa seguinte, *Set*, e desativam a etapa anterior, *Reset*, tornando este método mais facilmente percetível e menos extenso quando comparado com o método das equações lógicas.

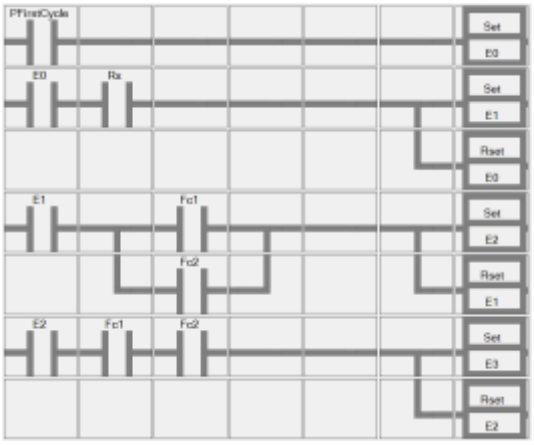


Figura 99 - Primeira parte das funções *Set/ Reset* do segundo exercício

Na Figura 99 observa-se a primeira parte do método das funções de *Set/ Reset*. Os dois primeiros degraus do projeto apenas são executados no primeiro ciclo do projeto ativando a "etapa 0" e posteriormente a "etapa 1" caso a entrada "Rx" seja ativada. No terceiro degrau, caso a "etapa 1" esteja ativa e um dos sensores de fim de curso, "fc1" ou "fc2", seja pressionado é ativada a "etapa 2" e desativada a "etapa 1".

Ainda na Figura 99, no quarto e último degrau, caso a "etapa 2" esteja ativa e ambos os sensores de fim de curso sejam ativados, esta etapa deixa de estar ativa passando a estar ativa a "etapa 3".

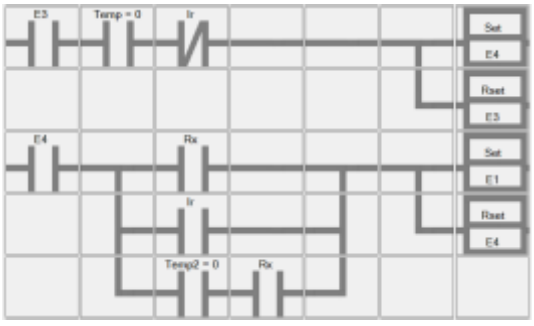


Figura 100 - Segunda parte das funções *Set/ Reset* do segundo exercício

Na Figura 100 observa-se a segunda parte das funções *Set/ Reset* do respetivo enunciado deste ponto. No quinto degrau caso a "etapa 3" esteja ativa, o temporizador chegar a 0 e a barreira infravermelhos não seja interrompida, esta etapa é desativada passando a estar ativa a "etapa 4".

No sexto degrau caso a "etapa 4" esteja ativa, correspondente ao fecho do portão e caso seja pressionado o "Rx"; ou a barreira de infravermelhos seja interrompida; ou o temporizador chegue a 0 e seja pressionado o "Rx", esta etapa passa a estar desativa e passa a estar ativa a "etapa 1" de abertura do portão.

Na figura 97 encontram-se os últimos três degraus que se referem às ativações das saídas nas respetivas etapas. Estes três degraus são iguais tanto no método de equações lógicas como no método de funções *Set/ Reset* pois, independentemente do método usado as saídas são ativadas da mesma forma.

Para ser possível programar o respetivo autómato programável, o código em *Ladder* é passado para C, podendo este ser encontrado no Apêndice J.

5.3 Semáforos

Neste terceiro exercício é pretendido um automatismo para comandar semáforos, como demostrado na Figura 101, em que um semáforo é designado por "A" e outro por "B", existindo um interruptor "IL" que permite ligar/desligar os semáforos.

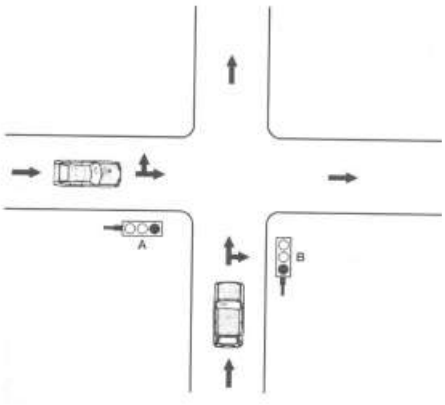


Figura 101 – Semáforos [Anexo A]

No modo desligado, o sinal amarelo é colocado em modo intermitente, em ambos os semáforos. No modo ligado o sinal vermelho e verde do semáforo oposto são ligados durante 30s, após este intervalo de tempo o sinal vermelho mantem-se ligado sendo desligado o sinal verde, no semáforo oposto e ligado o sinal amarelo durante 3s. Na passagem de amarelo para vermelho o

semáforo "A" e "B" ficam ambos com o sinal vermelho durante 1.5s voltando a repetir a sequência para o semáforo oposto. [Anexo A]

5.3.1. *Grafcet*

Recorrendo ao ambiente de desenvolvimento, é criado um projeto na língua *Grafcet* seguindo os requisitos pedidos no respetivo enunciado. O primeiro passo na criação de um projeto no ambiente de desenvolvimento é a criação da tabela de símbolos.

Nomes	Tipos	Posições	Comentarios
IL	Entrada	1	ligar/desligar semáforos
Timer	Temporizador	1	temporizador
Verm1	Saida	1	Vermelho 1ºsemáforo
Verm2	Saida	2	Vermelho 2ºsemáforo
Verde1	Saida	3	Verde 1ºsemáforo
Verde2	Saida	4	Verde 2ºsemáforo
Ama1	Saida	5	Amarelo 1ºsemáforo
Ama2	Saida	6	Amarelo 2ºsemáforo

Figura 102 - Tabela de símbolos *Grafcet* do terceiro exercício

Na Figura 102 observa-se a tabela de símbolos para a linguagem *Grafcet* do terceiro exercício. Nesta tabela são declaradas as variáveis pertencentes ao projeto, sendo usados três tipos de variáveis, do tipo entrada, saída e temporizador. Encontrando-se a tabela de símbolos definida é então possível iniciar a elaboração do código em *Grafcet*.

Figura 103 - Primeira parte do código em linguagem *Grafcet* do terceiro exercício

Na Figura 103 observa-se a primeira parte do código em linguagem *Grafcet* constituído pelas duas opções: ou os semáforos se encontram ligados, ou os semáforos se encontram desligados.

No primeiro ciclo do projeto a "etapa 1" fica ativa voltando a estar ativa quando a "etapa 3" se encontrar ativa e a transição "Timer=65" for valida ou a "etapa 9" se encontrar ativa e a transição "Timer=0" for valida.

Estando a "etapa 1" ativa, caso os semáforos estejam em modo desligado, "IL = 0", passa para a "etapa 2", caso estejam em modo ligado, "IL = 255" passa para a "etapa 4".

O modo desligado é constituído pelo ramo esquerdo do diagrama de *Grafcet* em que, na "etapa 2" são ligados os sinais luminosos amarelos em ambos os semáforos durante 3 segundos. Passados os 3 segundos a transição "Timer = 67" é validada passado a "etapa 3" a estar ativa onde são desligados todos os sinais luminosos durante 2 segundos. Após este intervalo de tempo, a transição "Timer = 65" é validada passando para a "etapa 1" e posteriormente para a "etapa 2" repetindo o ciclo enquanto os semáforos se encontrarem em modo desligado, "IL = 0".

O modo ligado é iniciado com a "etapa 4", em que o sinal luminoso vermelho do semáforo "A" e o sinal luminoso verde do semáforo "B" se encontram ligados durante 30 segundos. Concluindo o intervalo de tempo de 30 segundos a transição "Timer = 40" é validada passando a estar ativa a "etapa 5" onde o sinal luminoso verde do semáforo "B" é desligado passando a estar ligado o sinal luminoso amarelo no respetivo semáforo durante 3 segundos. Após os 3 segundos a transição "Timer = 37" é validada passando a estar ativa a "etapa 6" onde são ligados os sinais luminosos vermelhos em ambos os semáforos durante 2 segundos.

Na figura 104 observa-se a segunda e última parte do código em linguagem *Grafcet* que dá continuação à anterior. Após o intervalo de 2 segundos a transição "Timer = 35" é considerada válida passando a estar ativa a "etapa 7" onde o sinal luminoso verde do semáforo "A" e o sinal luminoso vermelho do semáforo "B" se encontram ligados durante 30 segundos. No fim deste intervalo de tempo de 30 segundos a transição "Timer = 5" é validada passando a estar ativa a "etapa" 8 onde o sinal luminoso vermelho do semáforo "B" se mantem ligado e o sinal luminoso verde do semáforo "A" é desligado passando ao estado ligado o sinal luminoso amarelo. Ambos os sinais luminosos mantém este estado durante 3 segundos. Decorrido este tempo ambos os sinais luminosos vermelhos os semáforos são ligados durante 2 segundos. Quando a transição "Timer = 0" for válida é concluído o ciclo voltando a repetir o ciclo enquanto se encontrar no modo ligado, "IL = 255".

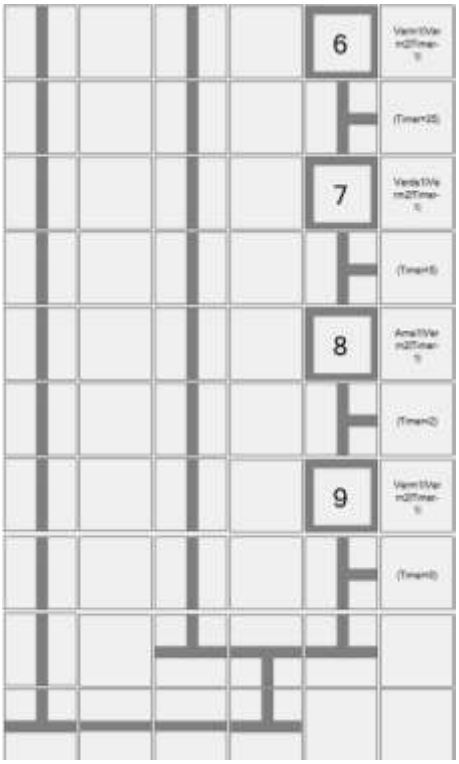


Figura 104 - Segunda parte do código em linguagem *Grafcet* do terceiro exercício

Concluindo o projeto, o código em *Grafcet* é passado para C por forma a ser possível programar o respetivo autómato programável, podendo este ser encontrado no Apêndice K.

5.3.2. *Ladder*

Iniciando um novo projeto no ambiente de desenvolvimento, na linguagem de *Ladder*, é possível elaborar o código do respetivo enunciado. Na Figura 105 observa-se a tabela de símbolos que descreve o nome das entradas/saídas/variáveis internas e temporizadores utilizados neste projeto, bem como as posições em que estas serão ligadas no respetivo autómato.

Nome	Tipo	Posição	Comentário
FFirstCycle	Interno	0	Primeiro Ciclo
IL	Entrada	1	ligar/desligar semáforos
Timer	Temporizador	1	temporizador
Verm1	Saída	1	Vermelho 1º semáforo
Verm2	Saída	2	Vermelho 2º semáforo
Verde1	Saída	3	Verde 1º semáforo
Verde2	Saída	4	Verde 2º semáforo
Ama1	Saída	5	Amarelo 1º semáforo
Ama2	Saída	6	Amarelo 2º semáforo
E0	Interno	1	etapa 0
E1	Interno	1	etapa 1
E2	Interno	1	etapa 2
E3	Interno	1	etapa 3
E4	Interno	1	etapa 4
E5	Interno	1	etapa 5
E6	Interno	1	etapa 6
E7	Interno	1	etapa 7
E8	Interno	1	etapa 8
E9	Interno	1	etapa 9

Figura 105 - Tabela de símbolos *Ladder* do terceiro exercício

Num projeto em *Ladder* é sempre necessário definir como variáveis internas as etapas do projeto a ser desenvolvido. A posição do temporizador não é considerada importante pois não tem nenhuma ligação ao exterior como as entradas e saídas do autómato.

Após a conclusão da tabela de símbolos é necessário elaborar o diagrama em *Ladder* que permitirá o funcionamento do automatismo, como pretendido no enunciado. No primeiro ponto proceder-se-á à elaboração através do método de equações lógicas e no segundo ponto através do método de funções *Set/ Reset*.

5.3.2.1. Método de Equações Lógicas

O primeiro passo no método de equações lógicas é definir as equações para o correto funcionamento do automatismo. No primeiro ciclo de execução do automatismo, a primeira equação lógica ativa a "etapa 0", sendo dada pela Equação (24). A segunda equação lógica tem como entrada todas as possibilidades da "etapa 1" ficar ativa, sendo dada pela Equação (25). Na terceira equação lógica são apresentadas todas as entradas que possibilitam a ativação da "etapa 2", sendo dada pelo Equação (26).

$$E_0 = P\_First\_Cycle \tag{24}$$

$$E_1 = (E_0 + E_1 + E_3 * Timer = 0 + E_9 * Timer = 0) * (\overline{E_2} + \overline{E_4}) \tag{25}$$

$$E_2 = (E_2 + E_1 * \overline{IL}) * \overline{E_3} \tag{26}$$

Na Figura 106 observa-se a primeira parte do código do diagrama *Ladder* correspondente às três primeiras equações: Equação (24), Equação (25) e Equação (26).

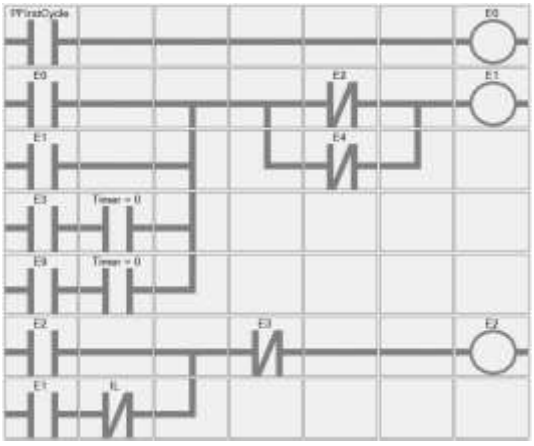


Figura 106 – Primeira parte das equações lógicas do terceiro exercício

As quarta, quinta, sexta, sétima, oitava, nova e décima, equações lógicas representam todas as entradas que, quando ativas, possibilitam a ativação da "etapa 3" à "etapa 9", respetivamente. Estas equações lógicas são dadas da Equação (27) à Equação (33), respetivamente.

$$E_3 = (E_3 + E_2 * Timer = 0) * \overline{E_1} \quad (27)$$

$$E_4 = (E_4 + E_1 * IL) * \overline{E_5} \quad (28)$$

$$E_5 = (E_5 + E_4 * Timer = 0) * \overline{E_6} \quad (29)$$

$$E_6 = (E_6 + E_5 * Timer = 0) * \overline{E_7} \quad (30)$$

$$E_7 = (E_7 + E_6 * Timer = 0) * \overline{E_8} \quad (31)$$

$$E_8 = (E_8 + E_7 * Timer = 0) * \overline{E_9} \quad (32)$$

$$E_9 = (E_9 + E_8 * Timer = 0) * \overline{E_1} \quad (33)$$

Na Figura 107 observa-se a segunda parte do código do diagrama *Ladder* correspondente às quatro primeiras equações acima referidas: da Equação (27) à Equação (30).

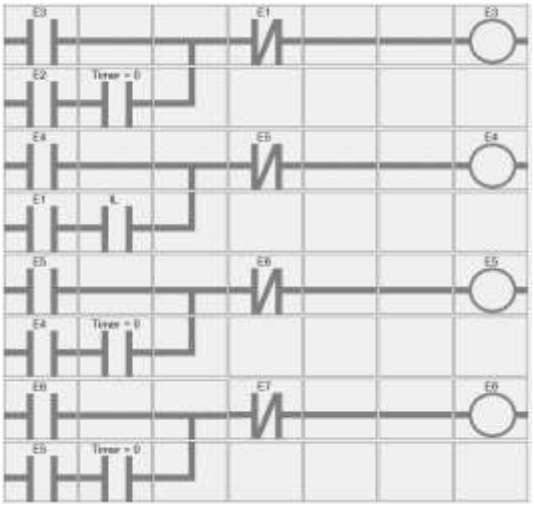


Figura 107 – Segunda parte das equações lógicas do terceiro exercício

Na Figura 108 observa-se a terceira parte do código do diagrama *Ladder* correspondente às três últimas equações acima referida: Equação (31), Equação (32) e Equação (33).

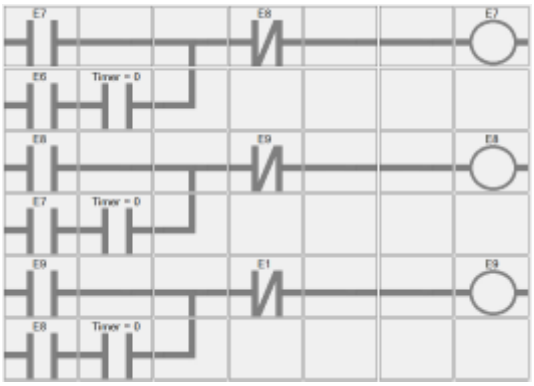


Figura 108 – Terceira parte das equações lógicas do terceiro exercício

A ativação das saídas acontece nas décima-primeira, décima-segunda, décima-terceira, décima quarta, décima-quinta, décima-sexta, décima-sétima e décima-oitava equações lógicas,

correspondendo às saídas ativas na “etapa 2”, “etapa 3”, “etapa 4”, “etapa 5”, “etapa 6”, “etapa 7”, “etapa 8” e “etapa 9”, respetivamente.

As equações abaixo, da Equação (34) à Equação (41), correspondem da décima-primeira à décima-oitava equações lógicas.

$$Ama1 + Ama2 + Timer - 1\ (\#3) = E_2 \quad (34)$$

$$Timer - 1\ (\#3) = E_3 \quad (35)$$

$$Verm1 + Verde2 + Timer - 1\ (\#30) = E_4 \quad (36)$$

$$Verm1 + Verm2 + Timer - 1\ (\#30) = E_5 \quad (37)$$

$$Verm1 + Verm2 + Timer - 1\ (\#2) = E_6 \quad (38)$$

$$Verm2 + Verde1 + Timer - 1\ (\#30) = E_7 \quad (39)$$

$$Ama1 + Verm2 + Timer - 1\ (\#3) = E_8 \quad (40)$$

$$Verm1 + Verm2 + Timer - 1\ (\#1) = E_9 \quad (41)$$

Na Figura 109 observa-se a quarta parte do código do diagrama *Ladder* correspondente às três primeiras equações acima referidas: Equação (34), Equação (35) e Equação (36).

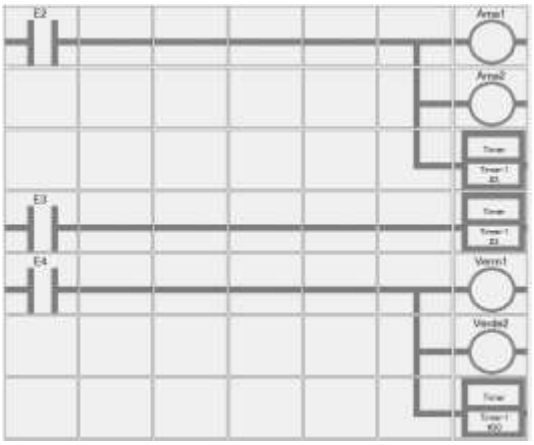


Figura 109 – Quarta parte das equações lógicas do terceiro exercício

Na Figura 110 observa-se a quinta parte do código do diagrama *Ladder* correspondente às seguintes duas equações acima referidas: Equação (37) e Equação (38).

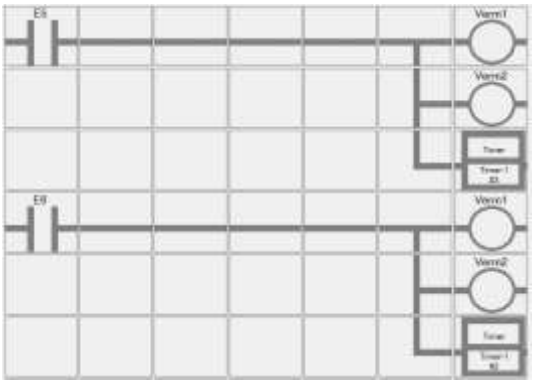


Figura 110 – Quinta parte das equações lógicas do terceiro exercício

Na Figura 111 observa-se a sexta parte do código do diagrama *Ladder* correspondente as três últimas equações acima referida: Equação (39), Equação (40) e Equação (41).

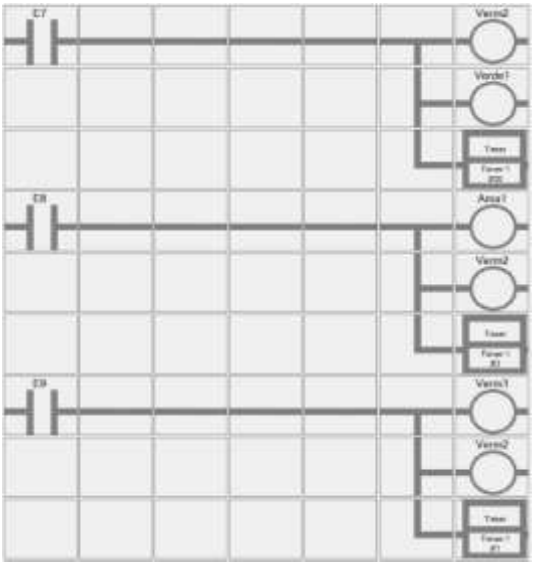


Figura 111 - Sexta parte das equações lógicas do terceiro exercício

As seguintes equações: décima-nona, vigésima, vigésima-primeira, vigésima-segunda, vigésima-terceira, vigésima-quarta, vigésima-quinta e vigésima-sexta, representam a descativação das etapas pertencentes ao projeto, correspondendo da Equação (42) à Equação (49), respetivamente.

$$\overline{E_0} + \overline{E_3} + \overline{E_9} = E_1 \quad (42)$$

$$\overline{E_1} = E_2 + E_4 \quad (43)$$

$$\overline{E_2} = E_3 \quad (44)$$

$$\overline{E_4} = E_5 \quad (45)$$

$$\overline{E_5} = E_6 \quad (46)$$

$$\overline{E_6} = E_7 \quad (47)$$

$$\overline{E_7} = E_8 \quad (48)$$

$$\overline{E_8} = E_9 \quad (49)$$

Na Figura 112 observa-se a sétima e última parte das equações lógicas onde estão representadas as oito últimas equações lógicas: Equação (42), Equação (43), Equação (44), Equação (45), Equação (46), Equação (47), Equação (48) e Equação (49).

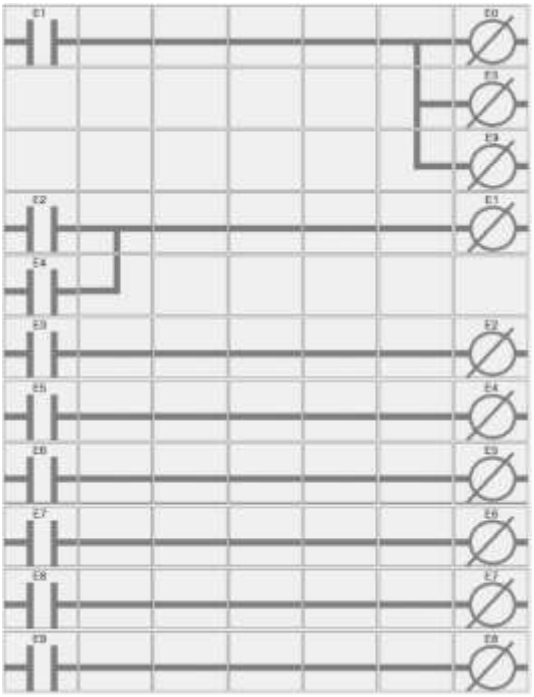


Figura 112 - Sétima parte das equações lógicas do terceiro exercício

Por fim, o código em *Ladder* é passado para C por forma a ser possível programar o respetivo autómato programável, podendo este ser encontrado no Apêndice L.

5.3.2.2. Método de Funções *Set/ Reset*

Partindo do diagrama de *Grafcet* e tendo a tabela de símbolos definida este método é de fácil implementação. As transições usadas no *Grafcet* são as equações que ativam a etapa seguinte, *Set*, e desativam a etapa anterior, *Reset*, tornando este método mais percetível e menos extenso, quando comparado com o método das equações lógicas.

Na Figura 113 observa-se a primeira parte do método das funções de *Set/ Reset*. Os dois primeiros degraus do projeto apenas são executados no primeiro ciclo do projeto, ativando a "etapa 0" e posteriormente a "etapa 1". No terceiro degrau, caso a "etapa 1" esteja ativa e "IL" esteja com o valor lógico zero é ativada a "etapa 2" e desativada a "etapa 1". N último degrau desta

parte, o quarto degrau, caso a "etapa 2" esteja ativa e o "Timer" for igual a zero esta etapa deixa de estar ativa passando a estar ativa a "etapa 3".

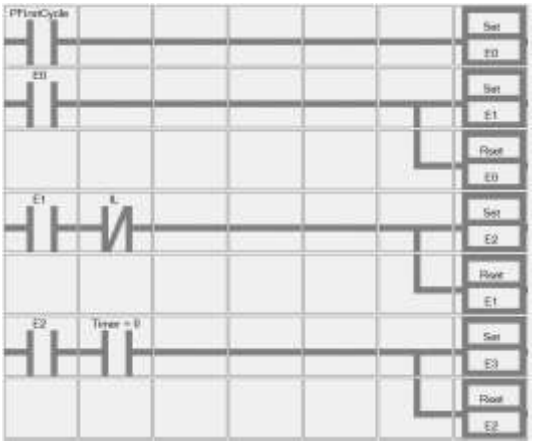


Figura 113 - Primeira parte das funções *Set/ Reset* do terceiro exercício

Na Figura 114 observa-se a segunda parte do método das funções de *Set/ Reset*. No primeiro degrau desta parte, o quinto degrau, caso a "etapa 3" esteja ativa e o "Timer" for igual a zero esta etapa deixa de estar ativa passando a estar ativa novamente a "etapa 1", iniciando novamente o ciclo de semáforos desligados.

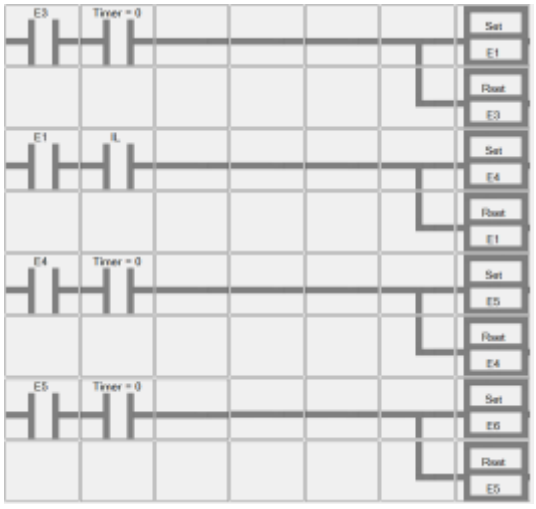


Figura 114 - Segunda parte das funções *Set/ Reset* do terceiro exercício

No modo ligado, no quinto degrau, caso a "etapa 1" esteja ativa e "IL" esteja com o valor lógico um, é ativada a "etapa 4" e desativada a "etapa 1". No sexto e sétimo degraus, caso a "etapa 4" e "etapa 5", respetivamente, estejam ativas e o "Timer" for igual a zero estas etapas deixam de estar ativas ficando ativas as "etapa 5" e "etapa 6", respetivamente.

Na Figura 115 observa-se a terceira parte do método das funções de *Set/ Reset*. No oitavo, nono, décimo e décimo-primeiro degrau, caso as "etapa 6", "etapa 7", "etapa 8" e "etapa 9",

respetivamente, estejam ativas e o “Timer” for igual a zero, estas etapas deixam de estar ativas. Neste caso ficam ativas as “etapa 7”, “etapa 8”, “etapa 9” e “etapa 1” respetivamente. Após a equação lógica do décimo-primeiro degrau ser validada é ativada novamente a “etapa 1”, iniciando novamente o ciclo de semáforos ligados.

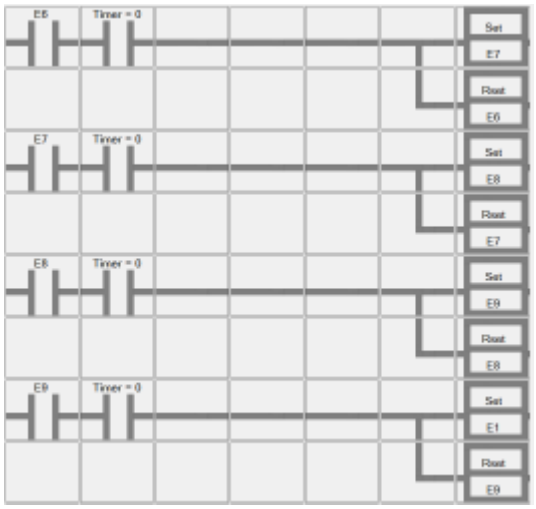


Figura 115 - Terceira parte das funções *Set/ Reset* do terceiro exercício

Nas figuras 109, 110 e 111 encontram-se os últimos degraus que se referem às ativações das saídas nas respetivas etapas- Estes degraus são iguais tanto no método de equações lógicas como no método de funções *Set/ Reset* pois, independentemente do método usado, as saídas são ativadas da mesma forma.

Para ser possível programar o respetivo autómato programável, o código em *Ladder* é passado para C, podendo ser encontrado no Apêndice M.

5.4 Transferência de Peças

Neste quarto exercício é pretendido um automatismo para transferir peças, entre dois tapetes com se pode observar na Figura 116.

Com recurso a dois botões de pressão, “s2” e “s0”, os tapetes são colocados em funcionamento ou paragem, respetivamente. Podendo a paragem ocorrer apenas no final de cada ciclo, existe um botão com encravamento, “s1”, para a imediata suspensão das ações em curso.

Quando pressionado o botão “s2”, ambos os tapetes entram em movimento, sendo transportadas peças no tapete 1 até ao sensor “s3” as detetar. Nesse momento o cilindro “A” avança empurrando-as para o tapete 2, finalizando com o recuo do cilindro “A”. No corpo dos

cilindros existem dois sensores magnéticos "a1" cilindro avançado e "a0" cilindro recuado. O ciclo repete-se sempre que "s3" detetar uma nova peça. [Anexo A]

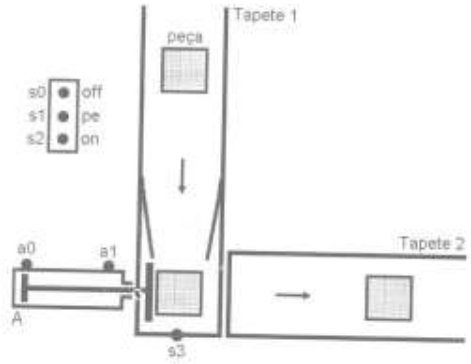


Figura 116 - Transferência de peças [Anexo A]

5.4.1. *Grafcet*

Utilizando o ambiente de desenvolvimento, é criado um projeto na linguagem *Grafcet* seguindo os requisitos pedidos no respetivo enunciado. A ter em atenção que o primeiro passo na criação de um projeto no ambiente de desenvolvimento é a criação da tabela de símbolos.

Nomes	Tipos	Posições	Comentarios
S0	Entrada	1	paragem
S1	Entrada	2	paragem de emergência
S2	Entrada	3	colocação em funcionamento
S3	Entrada	4	sensor que detecta
A1	Entrada	5	fim de curso do cilindro A+
A0	Entrada	6	fim de curso do cilindro A-
Afrente	Saida	1	cilindro A avança
Atras	Saida	2	cilindro A recua
KM1	Saida	3	ligar tapete 1
KM2	Saida	4	ligar tapete 2

Figura 117 - Tabela de símbolos *Grafcet* do quarto exercício

Na Figura 117 observa-se a tabela de símbolos para a linguagem *Grafcet* do quarto exercício. Nesta tabela são declaradas as variáveis pertencentes ao projeto e, neste caso, são apenas usadas variáveis do tipo entrada e saída. Encontrando-se a tabela de símbolos definida é então possível iniciar a elaboração do código em *Grafcet*.

Na Figura 118 observa-se a primeira parte do código em linguagem *Grafcet*, o automatismo pode encontrar-se na "etapa 1" de duas formas: no primeiro ciclo de execução do projeto e, sempre que der inicio a um novo ciclo. Para a "etapa 2" ficar ativa é necessário que seja pressionado o botão "s2", botão de colocação em funcionamento, e não estejam pressionados os botões "s0", botão de paragem, e "s1", botão de paragem de emergência. Na "etapa 2" são ativados os dois tapetes e, para a conclusão da referida etapa, existem duas opções: caso seja pressionado o botão "s1", botão de paragem de emergência, entra na "etapa 3" sendo

desativadas todas as saídas do projeto e permanecendo nesta etapa até ser libertado o respetivo botão; e, caso o botão “s1” não seja pressionado e o sensor “s3”, sensor que deteta as caixas, tenha o valor lógico um, é ativada a “etapa 4” prosseguindo o ciclo.

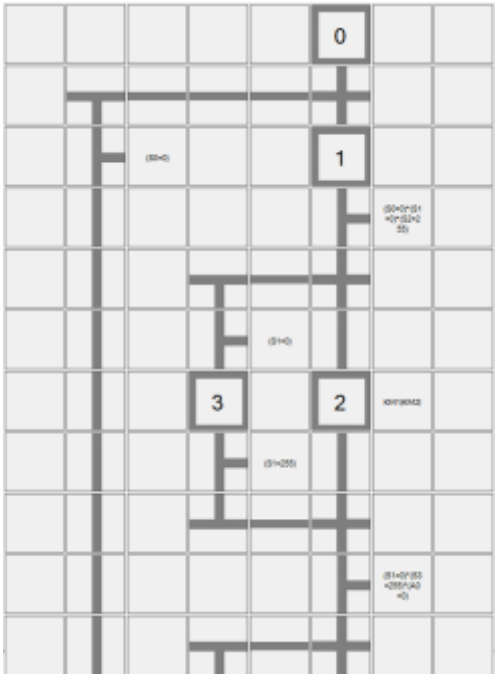


Figura 118 - Primeira parte do código em linguagem *Grafcet* do quarto exercício

Na Figura 119 observa-se a segunda e última parte do código em linguagem *Grafcet* que, continuando o ciclo, a “etapa 4” ativa saída correspondente novamente ao avanço do cilindro. Para a conclusão da referida etapa existem duas opções: uma correspondente à paragem de emergência que desativa todas as saídas enquanto o botão “s1” se mantiver pressionado; outra, caso o cilindro se encontre na posição mais avançada, ou seja, o sensor de fim de curso do cilindro avançado, “a1”, esteja pressionado. Neste segundo caso, o processo é passado para a “etapa 6” ou é ativada a saída correspondente ao recuo do cilindro, sendo este, dado como concluído caso o sensor de fim de curso do cilindro recuado, “a0”, tiver o valor lógico um. Neste caso o ciclo é novamente iniciado.

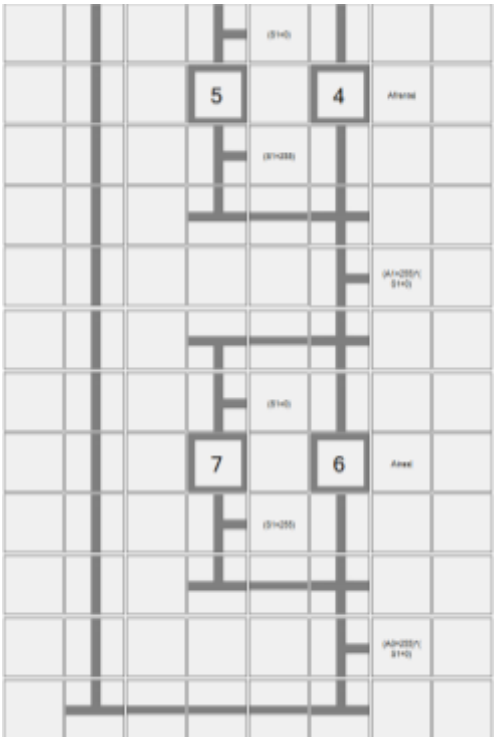


Figura 119 - Segunda parte do código em linguagem *Grafcet* do quarto exercício

Concluído o projeto, o código em *Grafcet* é passado para C por forma a ser possível programar o respetivo autómato programável, podendo este ser encontrado no Apêndice N.

5.4.2. *Ladder*

Iniciando um novo projeto no ambiente de desenvolvimento, na linguagem de *Ladder*, é possível elaborar o código do respetivo enunciado. Na Figura 120 observa-se a tabela de símbolos que descreve o nome das entradas/saídas e variáveis internas utilizadas neste projeto, bem como as posições em que estas serão ligadas no respetivo autómato. Mais uma vez e como já referido, num projeto em *Ladder* é sempre necessário definir como variáveis internas as etapas do projeto a ser desenvolvido.

Após a conclusão da tabela de símbolos é então necessário elaborar o diagrama em *Ladder*. O primeiro ponto vai ser elaborado através do método de equações lógicas e o segundo ponto vai ser elaborado através do método de funções *Set/ Reset*.

Nome	Tipo	Posição	Comentário
PFirstCycle	Interno	0	Primeiro Ciclo
S0	Entrada	1	paragem
S1	Entrada	2	paragem de emergência
S2	Entrada	3	colocação em funcionamento
S3	Entrada	4	sensor que deteta
A1	Entrada	5	fim do curso do cilindro Afrente
A0	Entrada	6	fim de curso do cilindro Atras
Afrente	Saída	1	cilindro A avança
Atras	Saída	2	cilindro A recua
KM1	Saída	3	ligar tapete 1
KM2	Saída	4	ligar tapete 2
E0	Interno	1	etapa 0
E1	Interno	1	etapa 1
E2	Interno	1	etapa 2
E3	Interno	1	etapa 3
E4	Interno	1	etapa 4
E5	Interno	1	etapa 5
E6	Interno	1	etapa 6
E7	Interno	1	etapa 7

Figura 120 - Tabela de símbolos *Ladder* do quarto exercício

5.4.2.1. Método de Equações Lógicas

O primeiro passo, no método de equações lógicas, é definir as equações para o correto funcionamento do automatismo. As equações lógicas englobam a ativação/desativação das etapas e saídas de todo o sistema.

No primeiro ciclo de execução do automatismo, a primeira equação lógica, ativa a "etapa 0", sendo dada pela Equação (50). As segunda, terceira, quarta, quinta, sexta, sétima e oitava equações lógicas representam todas as entradas que, quando ativas, possibilitam a ativação da "etapa 1" à "etapa 7", respetivamente. Estas equações lógicas são dadas da Equação (51) à Equação (57), respetivamente.

$$E_0 = P\_First\_Cycle \quad (50)$$

$$E_1 = (E_0 + E_1 + E_6 * A0 * \overline{S0} * \overline{S1}) * \overline{E_2} \quad (51)$$

$$E_2 = (E_2 + E_1 * \overline{S0} * \overline{S1} * S2 + E_3 * \overline{S1}) * \overline{E_4} \quad (52)$$

$$E_3 = (E_3 + E_2 * S1) * \overline{E_4} \quad (53)$$

$$E_4 = (E_4 + E_2 * \overline{S1} * S3 * \overline{A0} + E_5 * \overline{S1}) * \overline{E_6} \quad (54)$$

$$E_5 = (E_5 + E_4 * S1) * \overline{E_6} \quad (55)$$

$$E_6 = (E_6 + E_4 * A1 * \overline{S1} + E_7 * \overline{S1}) * \overline{E_1} \quad (56)$$

$$E_7 = (E_7 + E_6 * S1) * \overline{E_1} \quad (57)$$

Na Figura 121 observa-se a primeira parte do código do diagrama *Ladder* correspondente às três primeiras equações: Equação (50), Equação (51) e Equação (52).

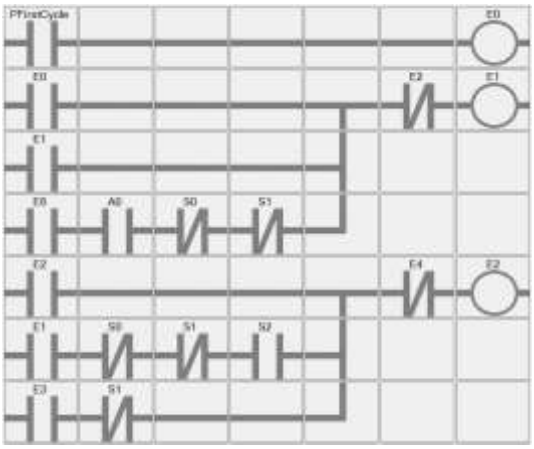


Figura 121 - Primeira parte das equações lógicas do quarto exercício

Na Figura 122 observa-se a segunda parte do código do diagrama *Ladder* correspondente às três seguintes equações: Equação (53), Equação (54) e Equação (55).

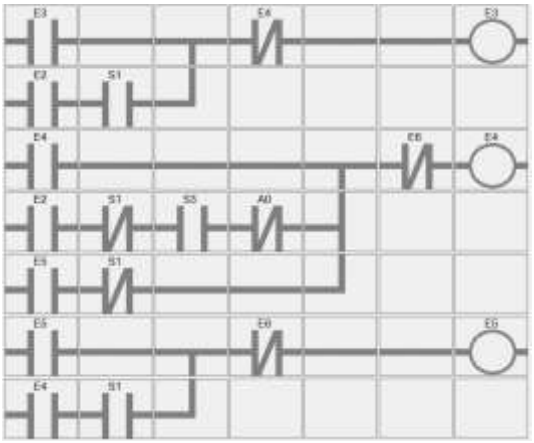


Figura 122 - Segunda parte das equações lógicas do quarto exercício

Na Figura 123 observa-se a terceira parte do código do diagrama *Ladder* correspondente às duas últimas equações de ativação de etapas: Equação (56) e Equação (57).

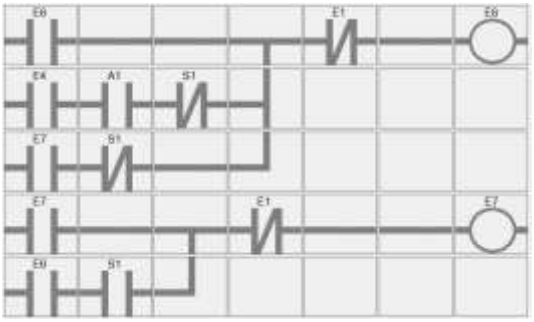


Figura 123 - Terceira parte das equações lógicas do quarto exercício

A ativação das saídas acontece na nona, décima e décima-primeira equações lógicas, correspondendo as saídas ativas na "etapa 2", "etapa 4" e "etapa 6", respetivamente. As

equações abaixo: Equação (58), Equação (59) e Equação (60) correspondem à nona, décima e décima-primeira equações lógicas.

$$KM1 + KM2 = E_2 \quad (58)$$

$$Afrente = E_4 \quad (59)$$

$$Atras = E_6 \quad (60)$$

Na Figura 124 observa-se a quarta parte do código do diagrama *Ladder* correspondente às três equações acima referidas: Equação (58), Equação (59) e Equação (60).

Figura 124 - Quarta parte das equações lógicas do quarto exercício

As seguintes equações: décima-segunda, décima-terceira, décima-quarta, décima-quinta, décima-sexta, décima-sétima, décima-oitava e décima-nona, representam a descativação das etapas pertencentes ao projeto, correspondendo da Equação (61) à Equação (68), respetivamente.

$$\overline{E_0} = E_1 \quad (61)$$

$$\overline{E_1} = E_2 \quad (62)$$

$$\overline{E_2} = E_3 + E_4 \quad (63)$$

$$\overline{E_3} = E_2 \quad (64)$$

$$\overline{E_4} = E_5 + E_6 \quad (65)$$

$$\overline{E_5} = E_4 \quad (66)$$

$$\overline{E_6} = E_1 + E_7 \quad (67)$$

$$\overline{E_7} = E_6 \quad (68)$$

Na Figura 125 observa-se a quinta, e última, parte das equações lógicas onde estão representadas as oito últimas equações lógicas: a Equação (61), Equação (62), Equação (63), Equação (64), Equação (65), Equação (66), Equação (67) e Equação (68).

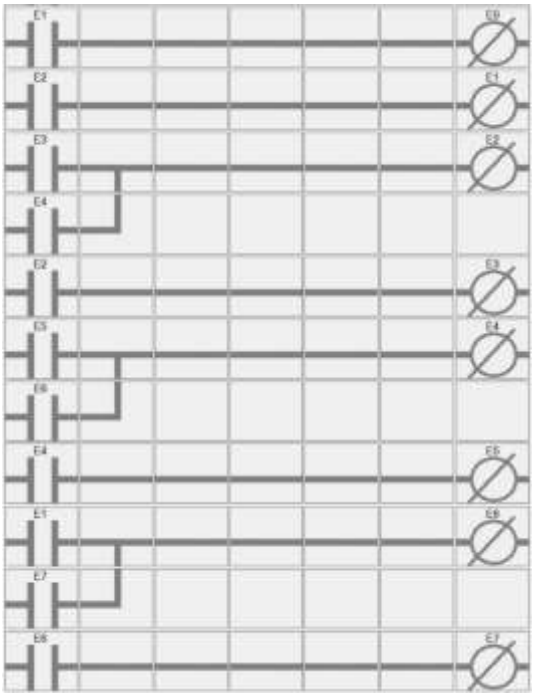


Figura 125 - Quinta parte das equações lógicas do quarto exercício

Por fim o código em *Ladder* é passado para C por forma a ser possível programar o respetivo autómato programável, podendo este ser encontrado no Apêndice O.

5.4.2.2. Método de Funções *Set/ Reset*

Partindo do diagrama de *Grafcet* e tendo a tabela de símbolos definida este método é de fácil implementação. As transições usadas no diagrama de *Grafcet* são as equações que ativam a etapa seguinte, *Set*, e desativam a etapa anterior, *Reset*, tornando este método mais percetível e menos extenso quando comparado com o método das equações lógicas.

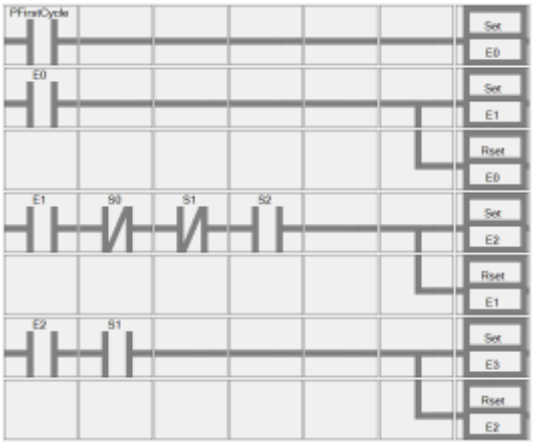


Figura 126 - Primeira parte das funções *Set/ Reset* do quarto exercício

Na Figura 126 observa-se a primeira parte do método das funções de *Set/ Reset*. Os dois primeiros degraus do projeto apenas são executados no primeiro ciclo do projeto ativando a "etapa

0” e posteriormente a “etapa 1”. No terceiro degrau caso a “etapa 1” esteja ativa, o botão “s0” e “s1” não sejam pressionados e o botão “s2” seja pressionado, é ativada a “etapa 2” e desativada a “etapa 1”. No último degrau, quarto degrau, caso a “etapa 2” esteja ativa e o botão “s1” seja pressionado, esta etapa deixa de estar ativa passando a estar ativa a “etapa 3”.

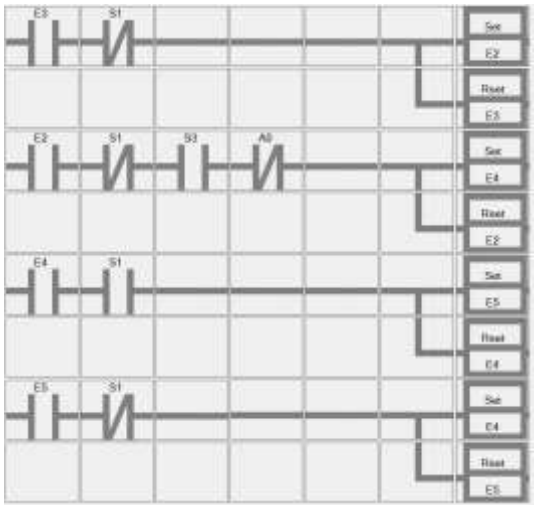


Figura 127 - Segunda parte das funções *Set/ Reset* do quarto exercício

Na Figura 127 observa-se a segunda parte do método das funções de *Set/ Reset*. No primeiro degrau, quinto degrau, caso a “etapa 3” esteja ativa e o botão “s1” não seja pressionado, esta etapa deixa de estar ativa passando novamente a estar ativa a “etapa 2”. No sexto degrau, caso a “etapa 2” esteja ativa, o botão “s1” não seja pressionado, o sensor “s3” detete uma caixa e o fim de curso do cilindro não esteja na posição avançada é ativada a “etapa 4” e desativa a “etapa 1”. Nos, sétimo e oitavo degraus, está representado, mais uma vez, o sistema de paragem de emergência. Estando, na “etapa 4” e caso o botão “s1” seja pressionado, é ativada a “etapa 5” e desativada a “etapa 4”. Volta à” etapa 4” quando deixar de se pressionar o botão “S1”.

Na Figura 128 observa-se a terceira parte do método das funções de *Set/ Reset*. No primeiro degrau desta parte, nono degrau, caso a “etapa 4” esteja ativa, o cilindro esteja na posição de avanço e o botão “s1” não seja pressionado, esta etapa deixa de estar ativa passando a estar ativa a “etapa 6”. Nos, décimo e décimo-primeiro degraus, é representado, mais uma vez, o sistema de paragem de emergência. Estando na “etapa 6” e, caso o botão “s1” seja pressionado, é ativada a “etapa 7” e desativada a “etapa 6”. Volta-se à “etapa 6” deixando de pressionar o botão “s1”. No último degrau desta terceira parte, caso a “etapa 6” esteja ativa, o cilindro esteja na posição de recuo e não estejam pressionados os botões “s1” e “s0” esta etapa deixa de estar ativa passando a estar ativa a “etapa 1”, recomeçando novamente o ciclo.

Na Figura 124 encontram-se os últimos degraus que se referem às ativações das saídas nas respetivas etapas. Estes degraus são iguais tanto no método de equações lógicas como no método de funções *Set/ Reset* pois, independentemente do método usado, as saídas são ativadas da mesma forma.

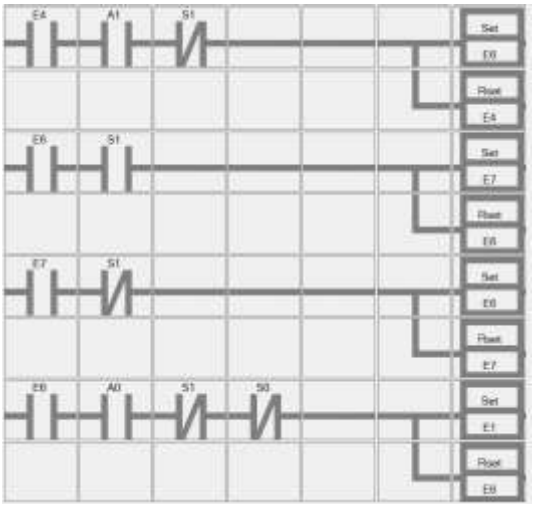


Figura 128 - Terceira parte das funções *Set/ Reset* do quarto exercício

Para ser possível programar o respetivo autómato programável, o código em *Ladder* é passado para C, podendo ser encontrado no Apêndice P.

5.5 Encaixotamento de Maçãs

Neste quinto exercício é pretendido um automatismo para controlar a linha de encaixotamento de maçãs, como se pode observar na Figura 129.

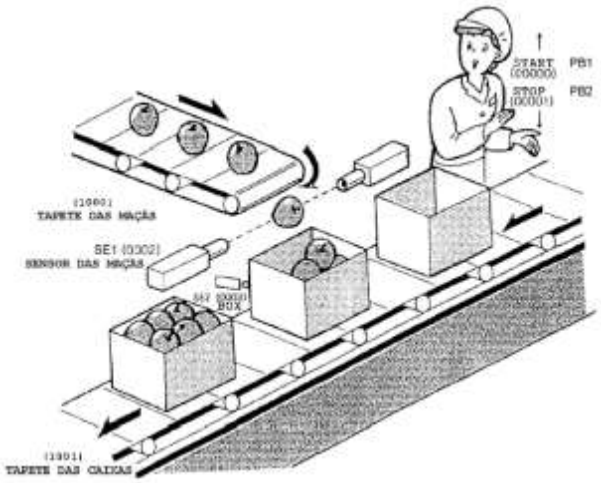


Figura 129 - Encaixotamento de maçaas [Anexo A]

O tapete das caixas entra em movimento quando o botão "Start" é pressionado. Quando o sensor das caixas "SE2" detetar uma caixa, o tapete da caixas é parado e colocado em funcionamento o tapete das maçãs. Quando o sensor das maças "SE1" deteta a passagem de 10 maças, o tapete das maças é parado e colocado novamente em funcionamento o tapete das caixas. O botão de "Stop" para o processo todo. [Anexo A]

5.5.1. *Grafcet*

Seguindo os requisitos pedidos no respetivo enunciado e utilizando o ambiente de desenvolvimento, é então criado um projeto na língua *Grafcet*. Como já foi sendo referido, o primeiro passo na criação de um projeto no ambiente de desenvolvimento é a criação da respetiva tabela de símbolos.

Nomes	Tipos	Posições	Comentarios
Start	Entrada	1	liga o sistema
SE1	Entrada	2	deteta maças
SE2	Entrada	3	deteta caixas
Stop	Entrada	4	para o sistema
Cnt	Interno	1	contador
M1	Saida	1	tapete das caixas
M2	Saida	2	tapete das maças

Figura 130 - Tabela de símbolos *Grafcet* do quinto exercício

Na Figura 130 observa-se a tabela de símbolos para a linguagem *Grafcet* do quinto exercício. Nesta tabela são declaradas as variáveis pertencentes ao projeto sendo, neste caso, apenas usadas variáveis do tipo entrada, saída e contador (interno). Encontrando-se a tabela de símbolos definida é então possível iniciar a elaboração do código em *Grafcet*.

Na Figura 131 observa-se o código em linguagem *Grafcet*. O automatismo pode encontrar-se na "etapa 1" de duas formas: no primeiro ciclo de execução do projeto e, sempre que der inicio a um novo ciclo completo. A "etapa 2" inicia o ciclo de encaixotamento de maçãs e, o automatismo pode encontrar-se ativo, igualmente, de duas formas: sempre que este se encontre na "etapa 1" e por pressionado o botão de "Start" ou sempre que o ciclo que encaixamento de maçãs recomeçar. Nesta etapa, o tapete das caixas é ativado.

A "etapa 3" é ativada caso o automatismo venha da "etapa 2" e o sensor "SE2" detete uma caixa. Neste caso é ativado o tapete das maçãs e iniciado o contador.

A "etapa 4" é ativada de duas formas: o automatismo encontra-se na "etapa 3" e o sensor "SE1" deteta a passagem da primeira maçã ou, o automatismo encontra-se na "etapa 5" e o sensor "SE1" deteta a passagem de uma maçã, não sendo esta a décima maçã da caixa. Nesta etapa continua ativo o tapete das maçãs e é incrementado o contador em uma unidade.

A "etapa 5" é ativada no ciclo seguinte à "etapa 4" estar ativa e mantém ativo o tapete das maçãs. Existem então três formas da "etapa 5" deixar de estar ativa: caso seja detetada outra maçã pelo sensor "SE1" e o contador for diferente de 10 maçãs, então volta à "etapa 4"; caso seja pressionado o botão de "Stop" voltando à "etapa 1" e, por fim, caso o contador chegue a 10, sendo ativada a "etapa 2" onde inicia um novo ciclo de encaixotamento de maçãs.

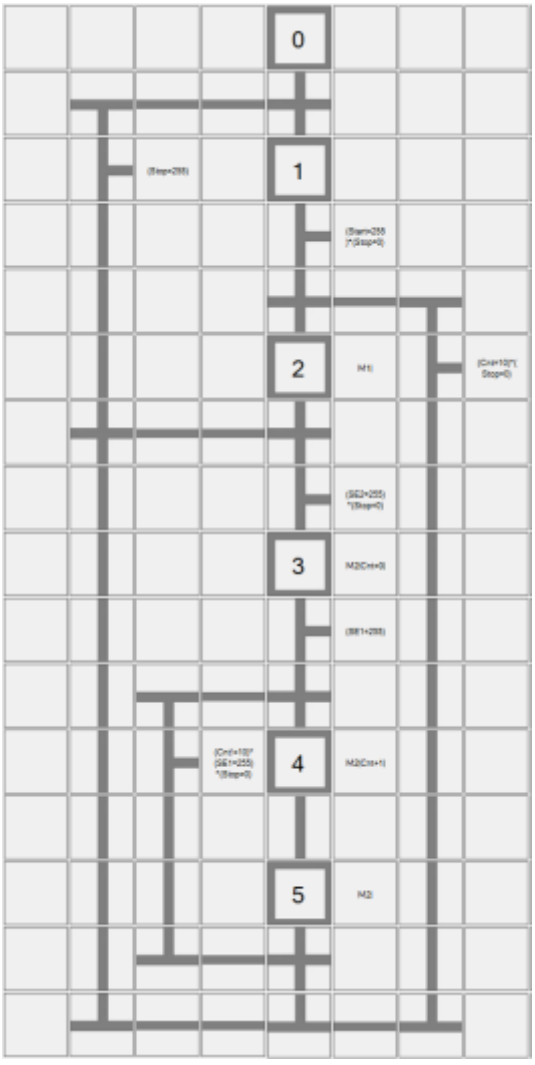


Figura 131 - Código em linguagem Grafcet do quinto exercício

Concluído o projeto, o código em *Grafcet* é passado para C por forma a ser possível programar o respetivo autómato programável, podendo este ser encontrado no Apêndice Q.

5.5.2. *Ladder*

Para um novo projeto no ambiente de desenvolvimento, na linguagem de *Ladder*, é possível elaborar o código do respetivo enunciado.

Nome	Tipo	Posição	Comentario
PFirstCycle	Interno	0	Primeiro Ciclo
Start	Entrada	1	liga o sistema
SE1	Entrada	2	deteta maçãs
SE2	Entrada	3	deteta caixas
Stop	Entrada	4	para o sistema
M1	Saida	1	tapete das caixas
M2	Saida	2	tapete das maçãs
CNT	Contador	1	contador de maçãs
E0	Interno	1	etapa 0
E1	Interno	1	etapa 1
E2	Interno	1	etapa 2
E3	Interno	1	etapa 3
E4	Interno	1	etapa 4
E5	Interno	1	etapa 5

Figura 132 - Tabela de símbolos *Ladder* do quinto exercício

Na Figura 132 observa-se a tabela de símbolos que descreve o nome das entradas/saídas e variáveis internas utilizadas neste projeto, bem como as posições em que estas serão ligadas no respetivo autómato. Como já referido, num projeto em *Ladder* é sempre necessário definir como variáveis internas as etapas do projeto a ser desenvolvido.

Após a conclusão da tabela de símbolos é então necessário elaborar o diagrama em *Ladder.* No primeiro ponto o procedimento vai ser elaborado através do método de equações lógicas e no segundo ponto através do método de funções *Set/ Reset*.

5.5.2.1. Método de Equações Lógicas

O primeiro passo no método de equações lógicas é definir as equações para o correto funcionamento do automatismo. As equações lógicas englobam a ativação/desativação das etapas e saídas de todo o sistema.

No primeiro ciclo de execução do automatismo, na primeira equação lógica, é ativa a" etapa 0", sendo dada pela Equação (69). As, segunda, terceira, quarta, quinta e sexta equações lógicas representam todas as entradas que, quando ativas possibilitam a ativação das, "etapa 1", "etapa 2", "etapa 3", "etapa 4" e "etapa 5", respetivamente. Estas equações lógicas são dadas pelas, Equação (70), Equação (71), Equação (72), Equação (73) e Equação (74), respetivamente.

$$E_0 = P\_First\_Cycle \quad (69)$$

$$E_1 = (E_1 + E_2 * Stop + E_5 * Stop) * \overline{E_2} \quad (70)$$

$$E_2 = (E_2 + E_1 * Start * \overline{Stop} + E_5 * CNT = 10 * \overline{Stop}) * \overline{E_3} \quad (71)$$

$$E_3 = (E_3 + E_2 * SE2 * \overline{Stop}) * \overline{E_4} \quad (72)$$

$$E_4 = (E_4 + E_3 * SE1 + E_5 * \overline{CNT = 10} * SE1 * \overline{Stop}) * \overline{E_1} \quad (73)$$

$$E_5 = (E_5 + E_4) * \overline{E_4} * \overline{E_1} \quad (74)$$

Na Figura 133 observa-se a primeira parte do código do diagrama *Ladder* correspondente às três primeiras equações: Equação (69), Equação (70) e Equação (71).

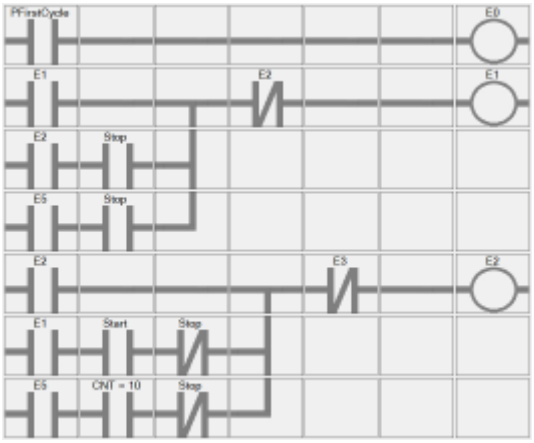


Figura 133 - Primeira parte das equações lógicas do quinto exercício

Na Figura 134 observa-se a segunda parte do código do diagrama *Ladder* correspondente às três equações seguintes: Equação (72), Equação (73) e Equação (74).

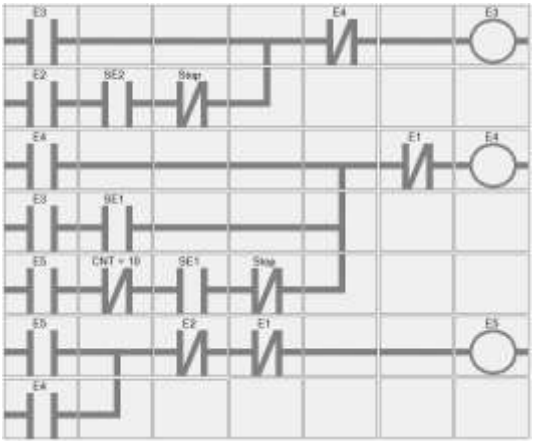


Figura 134 - Segunda parte das equações lógicas do quinto exercício

A ativação das saídas acontece nas, sétima, oitava e nona equações lógicas, correspondendo às saídas ativas na "etapa 2"; nas, "etapa 2", "etapa 4"e "etapa 5"; e na "etapa 3", respetivamente.

As equações abaixo: Equação (75), Equação (76) e Equação (77) correspondem às, sétima, oitava e nona equações lógicas.

$$M1 = E_2 \quad (75)$$

$$M2 = E_2 + E_4 + E_5 \quad (76)$$

$$CNT(\#0) = E_3 \quad (77)$$

A décima equação lógica diz respeito ao incremento do contador, sendo dado pela equação abaixo: Equação (78).

$$CNT(+1) = E_2 * SE1 \quad (78)$$

Na Figura 135 observa-se a terceira parte do código do diagrama *Ladder* correspondente às: quatro equações acima referidas, Equação (75), Equação (76), Equação (77) e Equação (78).

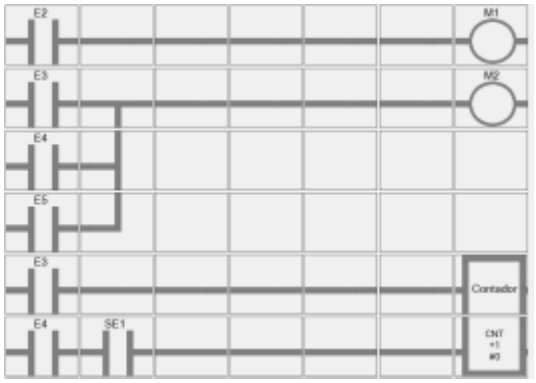


Figura 135 - Terceira parte das equações lógicas do quinto exercício

As seguintes equações: décima-primeira, décima-segunda, décima-terceira, décima-quarta, décima-quinta e décima-sexta, representam a descativação das etapas pertencentes ao projeto, correspondendo às, Equação (79), Equação (80), Equação (81), Equação (82), Equação (83) e Equação (84), respetivamente.

$$\overline{E_0} = E_1 \quad (79)$$

$$\overline{E_1} = E_2 \quad (80)$$

$$\overline{E_2} = E_1 + E_3 \quad (81)$$

$$\overline{E_3} = E_4 \quad (82)$$

$$\overline{E_4} = E_5 \quad (83)$$

$$\overline{E_5} = E_1 + E_2 \quad (84)$$

Na Figura 136 observa-se a quarta e última parte das equações lógicas, onde estão representadas as seis últimas equações lógicas: Equação (79), Equação (80), Equação (81), Equação (82), Equação (83) e Equação (84).

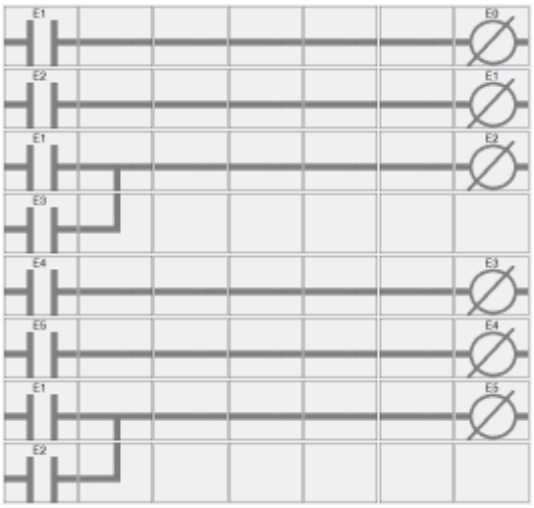


Figura 136 - Quarta parte das equações lógicas do quinto exercício

Por fim o código em *Ladder* é passado para C por forma a ser possível programar o respetivo autómato programável, podendo este ser encontrado no Apêndice R.

5.5.2.2. Método de Funções *Set/ Reset*

Partindo do diagrama de *Grafcet* e tendo a tabela de símbolos definida este método é de fácil implementação. As transições usadas no diagrama de *Grafcet* são as equações que ativam a etapa seguinte, *Set*, e desativam a etapa anterior, *Reset*, tornando este método mais percetível e menos extenso, quando comparado com o método das equações lógicas.

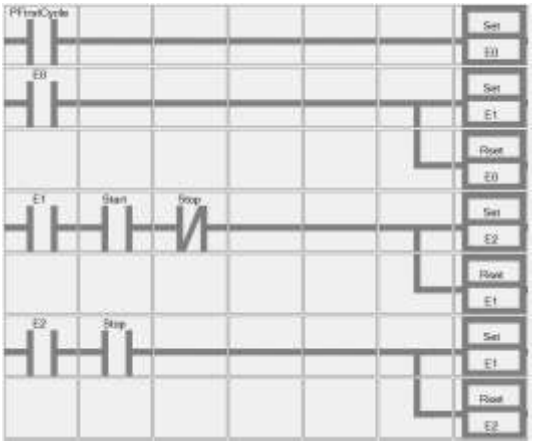


Figura 137 - Primeira parte das funções *Set/ Reset* do quinto exercício

Na Figura 137 observa-se a primeira parte do método das funções de *Set/ Reset*. Os dois primeiros degraus do projeto apenas são executados no primeiro ciclo do projeto, ativando a "etapa 0" e posteriormente a "etapa 1". No terceiro degrau, caso a "etapa 1" esteja ativa, o botão "Stop" não seja pressionado e o botão "Start" seja pressionado, é ativada a "etapa 2" e desativada a "etapa 1". No último degrau desta parte, quarto degrau, caso a "etapa 2" esteja ativa e o botão

"Stop" seja pressionado, esta etapa deixa de estar ativa passando a estar ativa a "etapa 1" novamente.

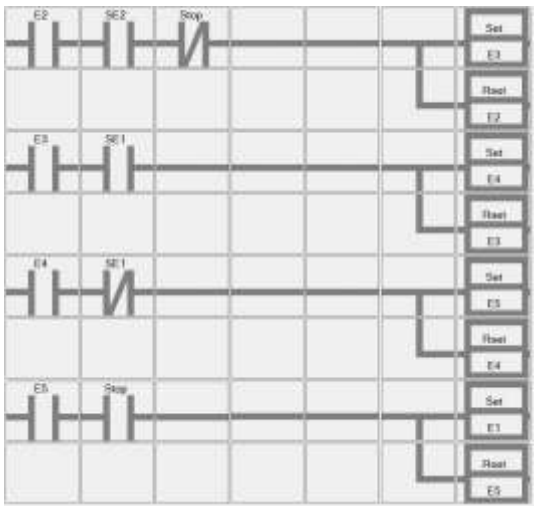


Figura 138 - Segunda parte das funções *Set/ Reset* do quinto exercício

Na Figura 138 observa-se a segunda parte do método das funções de *Set/ Reset*. No primeiro degrau desta parte, quinto degrau, caso a "etapa 2" esteja ativa, o botão "Stop" não seja pressionado e o sensor "SE2" detete uma caixa, esta etapa deixa de estar ativa passando a estar ativa a "etapa 3". No sexto degrau, caso a "etapa 3" esteja ativa e o sensor "SE1" detete uma maçã, passa a estar ativa a "etapa 4" e é desativada a "etapa 3". No sétimo degrau, caso a "etapa 4" esteja ativa e deixe de detetar uma maçã no sensor "SE1", é ativada a "etapa 5" e desativada a "etapa 4". No último degrau desta parte, oitavo degrau, caso a "etapa 5" esteja ativa e seja pressionado o botão de "Stop", o automatismo desativa a "etapa 5", ativando a primeira etapa do projeto, "etapa 1".

Figura 139 - Terceira parte das funções *Set/ Reset* do quinto exercício

Na Figura 139 observa-se a terceira parte do método das funções de *Set/ Reset*. No primeiro degrau desta parte, nono degrau, caso a "etapa 5" esteja ativa, o contador não tenha chegado ao valor 10, o sensor "SE1" detete novamente uma maçã e não seja pressionado o botão de "Stop", esta etapa deixa de estar ativa passando a estar ativa a "etapa 4", novamente. No último degrau desta parte, décimo degrau, caso novamente a "etapa 5" esteja ativa, o contador chegue ao valor

10 e não seja pressionado o botão de "Stop", é ativada a "etapa 2", iniciando um novo ciclo de encaixotamento de maçãs.

Na Figura 135 encontram-se os últimos degraus que se referem às ativações das saídas nas respetivas etapas. Estes degraus são iguais tanto no método de equações lógicas como no método de funções *Set/ Reset* pois, independentemente do método usado, as saídas são ativadas da mesma forma.

Para ser possível programar o respetivo autómato programável o código em *Ladder* é passado para C, podendo ser encontrado no Apêndice S.

5.6 Pesagem e Mistura

Neste sexto e último exercício pretende-se um automatismo de pesagem e mistura, como se pode observar na Figura 140. Na balança "C" são pesados os dois produtos: "A" e "B", separadamente através da abertura das válvulas "VA" e "VB", respetivamente. Após a pesagem, é aberta a válvula "VC" despejando na misturadora "N".

Os briquetes, igualmente misturados, são transportados por um tapete rolante, acionado pelo motor "MT", até à misturadora "N". A contagem dos briquetes é feita através de um detetor de passagem "DP".

A mistura é composta por uma quantidade "a" do produto "A", uma quantidade "b" do produto "B" e dois briquetes. A mistura é feita com recurso ao motor de rotação "MR" durante 15 segundos. Concluído o ciclo, a mistura é despejada através do motor que pilota o despejo, "MP" existindo um fim de curso, parando o motor na posição desejada. O processo inicia-se após ser pressionado o botão de encravamento "DCy". [Anexo A]

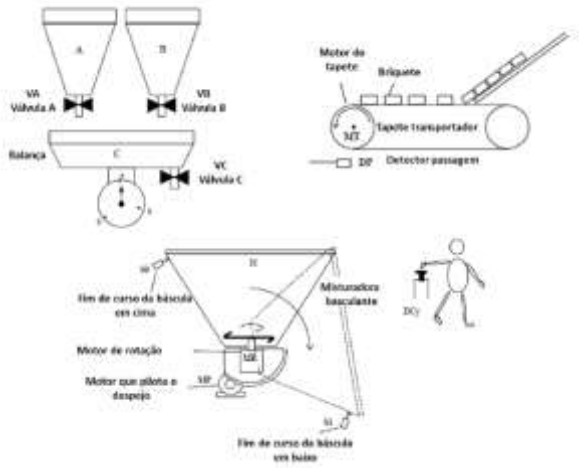


Figura 140 - Pesagem e mistura [Anexo A]

5.6.1. *Grafcet*

Recorrendo ao ambiente de desenvolvimento, foi elaborado o projeto na linguagem de *Grafcet* do respetivo enunciado. Na Figura 141 observa-se a tabela de símbolos que descreve o nome das entradas/saídas/temporizadores utilizados neste projeto, bem como as posições em que estes serão ligadas no respetivo autómato. A posição do temporizador usado neste projeto não é relevante.

Nomes	Tipos	Posições	Comentarios
Dcy	Entrada	1	inicio do processo
Dp	Entrada	2	detetor de passagem
B	Entrada	3	balança
S0	Entrada	4	fim de curso cima
S1	Entrada	5	fim de curso baixo
Timer	Temporizador	1	temporizador
Va	Saida	1	válvula A
Vb	Saida	2	válvula B
Vc	Saida	3	válvula C
Mt	Saida	4	motor tapete
Mr	Saida	5	motor de rotação
Mp	Saida	6	motor que pilota o despejo

Figura 141 - Tabela de símbolos *Grafcet* do sexto exercício

Estando, então, a tabela de símbolos bem definida é necessário elaborar o diagrama em *Grafcet* que permita o funcionamento do automatismo, como pretendido no enunciado. Assim, pode observar-se na Figura 142 a primeira parte do código em linguagem *Grafcet,* utilizando uma estrutura lógica AND que permite a execução simultânea de sequências paralelas.

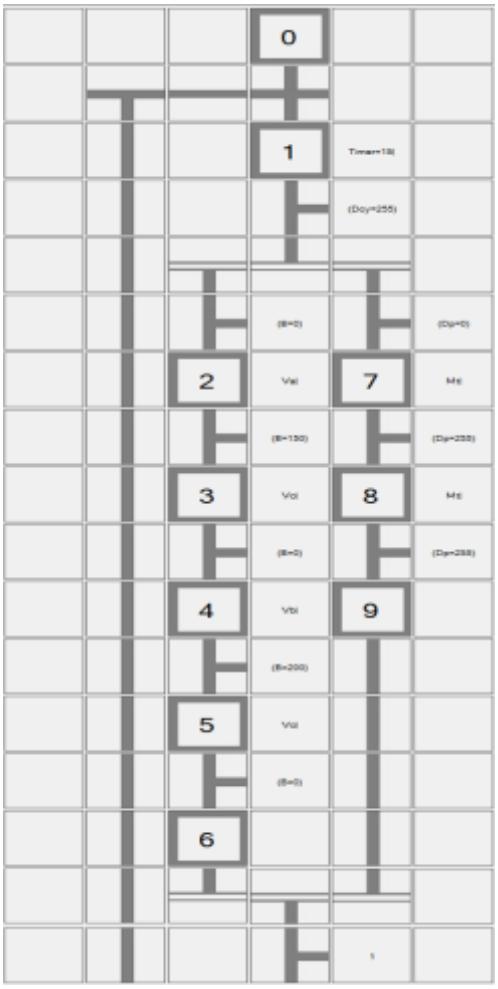


Figura 142 - Primeira parte do código em linguagem *Grafcet* do sexto exercício

A "etapa 1", inicio de um novo ciclo, quando ativa, inicializa o temporizador a 15. Com esta etapa ativa, caso o botão de encravamento "Dcy" seja pressionado são ativadas simultaneamente as "etapa 2" e "etapa 7" caso o valor da balança "B" esteja a zero e o detetor de passagem "DP" esteja também a zero. A "etapa 2" e a "etapa 7", estando ativas em simultâneo, iniciam a execução, igualmente simultânea, das respetivas sequências.

Na "etapa 2" é aberta a válvula "VA" do produto "A" até que o valor da balança "B" seja igual à quantidade desejada do produto "A". Atingida a quantidade do produto "A", é ativada a "etapa 3" sendo aberta a válvula "VC" e despejando o respetivo produto na misturadora "N". Quando o valor da balança "B" chegar a zero significa que o despejar terminou, ativando a "etapa 4".

Na "etapa 4" é aberta a válvula "VB" do produto "B" até que o valor da balança "B" seja igual à quantidade desejada do produto "B". Alcançada a quantidade do produto "B", é ativada a "etapa 5" sendo aberta a válvula "VC" e despejando o respetivo produto na misturadora "N". Quando o valor da balança "B" chegar a zero significa que o despejar terminou, ativando a "etapa 6".

Na “etapa 7” é ativado o motor do tapete que transporta os briquetes. Caso seja detetado um briquete através do detetor de passagem “DP” é ativada a “etapa 8” que tem o comportamento igual ao da “etapa 7”, ativando a etapa seguinte, “etapa 9”.

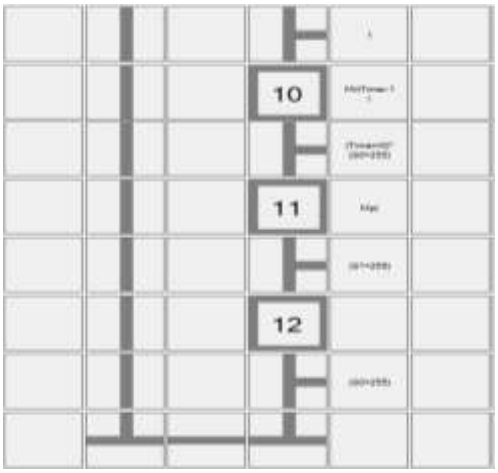


Figura 143 - Segunda parte do código em linguagem *Grafcet* do sexto exercício

Encontrando-se o automatismo com as, “etapa 6” e “etapa 9”, ativas é então ativada a “etapa 10”. Na Figura 143 pode observar-se o correspondente à segunda parte do código em linguagem *Grafcet*.

Na “etapa 10” é ativado o motor de rotação “MR” que mistura a solução durante 15 segundos. Terminando esse intervalo de tempo e, caso a mistura “N” se encontre na posição superior, ou seja, o sensor de fim de curso superior “S0” esteja a ser pressionado, é ativada a “etapa 11”.

Na “etapa 11” é ativado o motor que pilota o despejo “MP” até que o fim de curso inferior “S1” seja pressionado. Neste caso é ativada a “etapa 12”, deixando de atuar o motor até que a misturadora “N” volte à posição superior e o sensor de fim de curso superior “S0” seja pressionado, sendo então ativada a “etapa 1” e iniciando novamente o ciclo.

Concluindo o projeto, o código em *Grafcet* é passado para C, por forma a ser possível programar o respetivo autómato programável, podendo este ser encontrado no apêndice T.

5.6.2. *Ladder*

No ambiente de desenvolvimento foi elaborado o projeto na linguagem de *Ladder* do respetivo enunciado. Na Figura 144 observa-se a tabela de símbolos que descreve o nome das entradas/saídas/variáveis internas utilizadas neste projeto, bem como as posições em que estas serão ligadas no respetivo autómato. Mais uma vez é de referir que, num projeto em *Ladder* é sempre necessário definir como variáveis internas as etapas do projeto a ser desenvolvido. As seis

últimas variáveis desta tabela são automaticamente adicionadas a partir do momento que se adiciona, ao projeto, um comparador. Três destas variáveis comparam o valor "B" com "a" e as outras três comparam o valor "B" com "b".

Nome	Tipo	Posição	Comentario
FirstCycle	Interno	0	Primeiro Ciclo
Dcy	Entrada	1	inicio do processo
Dp	Entrada	2	detetor de passagem
B	Entrada	3	balança
S0	Entrada	4	fim de curso cima
S1	Entrada	5	fim de curso baixo
Temp	Temporizador	1	temporizador
Va	Saida	1	válvula A
Vb	Saida	2	válvula B
Vc	Saida	3	válvula C
Mt	Saida	4	motor tapete
Mr	Saida	5	motor de rotação
Mp	Saida	6	motor que pilota o despejo
a	Interno	1	quantidade do produto A
b	Interno	1	quantidade do produto B
E0	Interno	1	etapa 0
E1	Interno	1	etapa 1
E2	Interno	1	etapa 2
E3	Interno	1	etapa 3
E4	Interno	1	etapa 4
E5	Interno	1	etapa 5
E6	Interno	1	etapa 6
E7	Interno	1	etapa 7
E8	Interno	1	etapa 8
E9	Interno	1	etapa 9
E10	Interno	1	etapa 10
E11	Interno	1	etapa 11
E12	Interno	1	etapa 12
BMenorQuea	Interno	1	B é menor que a
BigualAa	Interno	1	B é igual a a
BMaiorQuea	Interno	1	B é maior que a
BMenorQueb	Interno	1	B é menor que b
BigualAb	Interno	1	B é igual a b
BMaiorQueb	Interno	1	B é maior que b

Figura 144 - Tabela de símbolos *Ladder* do sexto exercício

Após a conclusão da tabela de símbolos é necessário elaborar o diagrama em *Ladder*. O primeiro ponto vai ser elaborado através do método de equações lógicas e o segundo ponto através do método de funções *Set/ Reset*.

5.6.2.1. Método de Equações Lógicas

O primeiro passo, no método de equações lógicas, é definir as equações para o correto funcionamento do automatismo. As equações lógicas englobam a ativação/desativação das etapas e saídas de todo o sistema.

No primeiro ciclo de execução do automatismo, na primeira equação lógica, é ativada a "etapa 0", sendo dada pela Equação (85). As, segunda, terceira, quarta, quinta, sexta, sétima, oitava, décima, décima-primeira, décima-segunda e décima-terceira, equações lógicas representam todas as entradas que, quando ativas, possibilitam a ativação das, "etapa 1" à "etapa 12", respetivamente.

Nas, quarta e sexta, equações lógicas existem duas entradas em cada equação que dependem do comparador. A quarta equação tem duas entradas do comparador: uma para o valor da balança "B" ser maior e outra para ser igual à quantidade pretendida do produto "A". Na sexta equação existem outras duas entradas do comparador: uma para o valor da balança "B" ser maior e outra para ser igual à quantidade pretendida do produto "B".

Da segunda à décima-terceira, equações lógicas são dadas das, Equação (86) à Equação (97), respetivamente.

$$E_0 = P\_First\_Cycle \quad (85)$$

$$E_1 = (E_0 + E_1 + E_{12} * S0) * \overline{E_2} * \overline{E_7} \quad (86)$$

$$E_2 = (E_2 + E_1 * Dcy * B = 0) * \overline{E_3} \quad (87)$$

$$E_3 = (E_3 + E_2 * (B > a + B = a)) * \overline{E_4} \quad (88)$$

$$E_4 = (E_4 + E_3 * B = 0) * \overline{E_5} \quad (89)$$

$$E_5 = (E_5 + E_4 * (B > b + B = b)) * \overline{E_6} \quad (90)$$

$$E_6 = (E_6 + E_5 * B = 0) * \overline{E_{10}} \quad (91)$$

$$E_7 = (E_7 + E_1 * Dcy * \overline{DP}) * \overline{E_8} \quad (92)$$

$$E_8 = (E_8 + E_7 * DP) * \overline{E_9} \quad (93)$$

$$E_9 = (E_9 + E_8 * DP) * \overline{E_{10}} \quad (94)$$

$$E_{10} = (E_{10} + E_6 * E_9) * \overline{E_{11}} \quad (95)$$

$$E_{11} = (E_{11} + E_{10} * Temp = 0 * S0) * \overline{E_{12}} \quad (96)$$

$$E_{12} = (E_{12} + E_{11} * S1) * \overline{E_1} \quad (97)$$

Na Figura 145 observa-se a primeira parte do código do diagrama *Ladder* correspondente às quatro primeiras equações: Equação (85), Equação (86), Equação (87) e Equação (88).

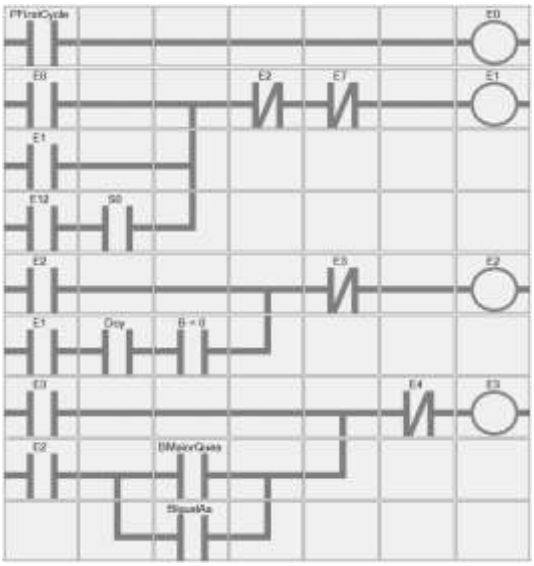


Figura 145 - Primeira parte das equações lógicas do sexto exercício

Na Figura 146 observa-se a segunda parte do código do diagrama *Ladder* correspondente às quatro seguintes equações: Equação (89), Equação (90), Equação (91) e Equação (92).

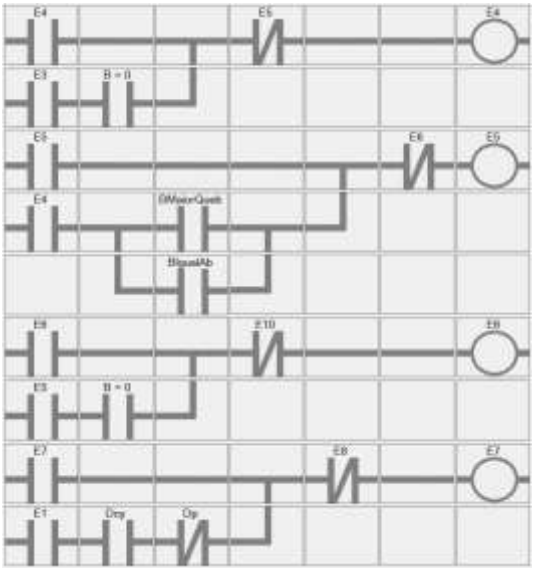


Figura 146 - Segunda parte das equações lógicas do sexto exercício

Na Figura 147 observa-se a terceira parte do código do diagrama *Ladder* correspondente às cinco seguintes equações: da Equação (93) à Equação (97).

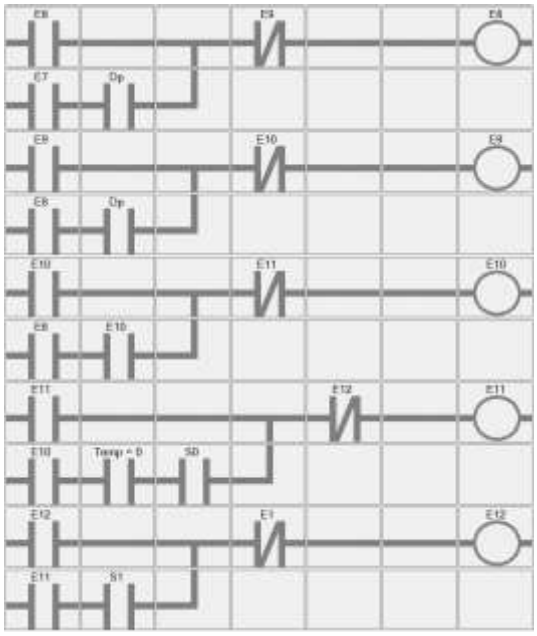


Figura 147 - Terceira parte das equações lógicas do sexto exercício

A ativação das saídas acontece nas, décima-quarta, décima-quinta, décima-sexta, décima-sétima, décima-oitava e décima-nona, equações lógicas, correspondendo às saídas ativas na "etapa 2"; na "etapa 2" e "etapa 5"; na "etapa 4"; na "etapa 7" e "etapa 8"; na "etapa 10"; e na "etapa 11", respetivamente.

Na décima-quarta e na décima-sexta equações lógicas são ativados dois comparadores, não se obtendo saídas físicas. Um comparador compara "B" com a quantidade pretendida do produto "A" e o outro, compara "B" com a quantidade pretendida do produto "B", respetivamente. Na décima-oitava equação lógica é ativado um temporizador que decrementa uma unidade, sendo iniciado a 15 e não correspondendo a uma saída física.

As seguintes equações, da Equação (98) à Equação (103), correspondem às, décima-quarta, décima-quinta, décima-sexta, décima-sétima, décima-oitava e décima-nona, equações lógicas.

$$Va + Comp(B, a) = E_2 \quad (98)$$

$$Vc = E_2 + E_5 \quad (99)$$

$$Vb + Comp(B, b) = E_4 \quad (100)$$

$$Mt = E_7 + E_8 \quad (101)$$

$$Mr + Timer - 1(\#15) = E_{10} \quad (102)$$

$$Mp = E_{11} \quad (103)$$

Na Figura 148 observa-se a quarta parte do código do diagrama *Ladder* correspondente às seis equações acima referidas: da Equação (98) à Equação (103).

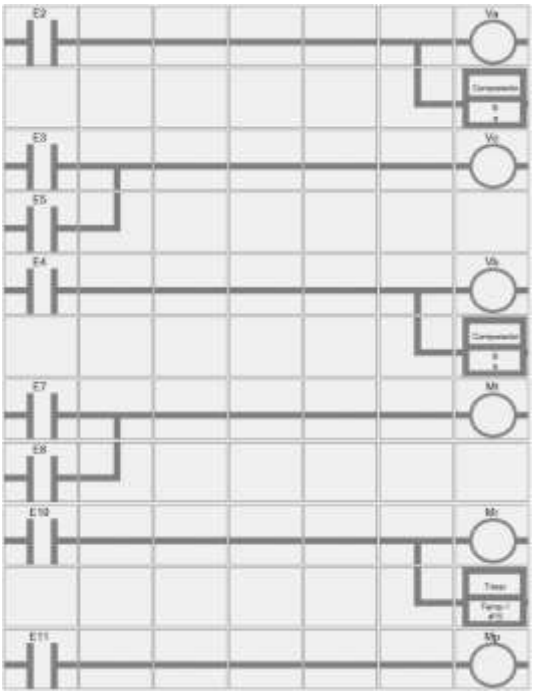


Figura 148 - Quarta parte das equações lógicas do sexto exercício

As últimas equações: vigésima, vigésima-primeira, vigésima-segunda, vigésima-terceira, vigésima-quarta, vigésima-quinta, vigésima-sexta, vigésima-sétima, vigésima-oitava, vigésima-nona, trigésima, trigésima-primeira e trigésima-segunda, representam a descativação das etapas pertencentes ao projeto, correspondendo: da Equação (104) à Equação (116), respetivamente.

$$\overline{E_0} = E_1 \quad (104)$$

$$\overline{E_1} = E_2 * E_7 \quad (105)$$

$$\overline{E_2} = E_3 \quad (106)$$

$$\overline{E_3} = E_4 \quad (107)$$

$$\overline{E_4} = E_5 \quad (108)$$

$$\overline{E_5} = E_6 \quad (109)$$

$$\overline{E_6} = E_7 \quad (110)$$

$$\overline{E_7} = E_8 \quad (111)$$

$$\overline{E_8} = E_9 \quad (112)$$

$$\overline{E_9} = E_{10} \quad (113)$$

$$\overline{E_{10}} = E_{11} \quad (114)$$

$$\overline{E_{11}} = E_{12} \quad (115)$$

$$\overline{E_{12}} = E_1 \quad (116)$$

Na Figura 149 observa-se a quinta e última parte das equações lógicas, onde estão representadas as treze últimas equações: da Equação (104) até à Equação (116).

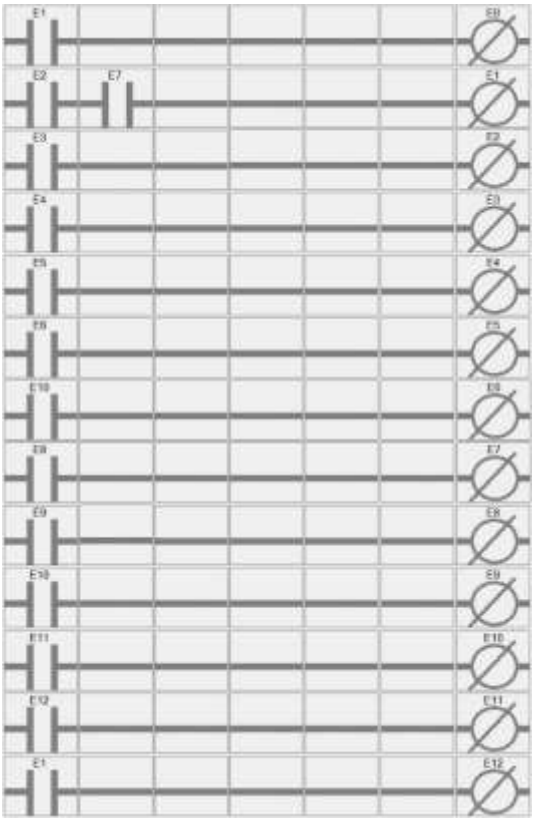


Figura 149 - Quinta parte das equações lógicas do sexto exercício

Por fim o código em *Ladder* é passado para C por forma a ser possível programar o respetivo autómato programável, podendo ser encontrado no Apêndice U.

5.6.2.2. Método de Funções *Set/ Reset*

Partindo do diagrama de *Grafcet* e tendo a tabela de símbolos definida este método é de fácil implementação. Nas transições usadas no diagrama de *Grafcet* são as equações que ativam a etapa seguinte, *Set*, e desativam a etapa anterior, *Reset*, tornando este método mais percetível e menos extenso, quando comparado com o método das equações lógicas.

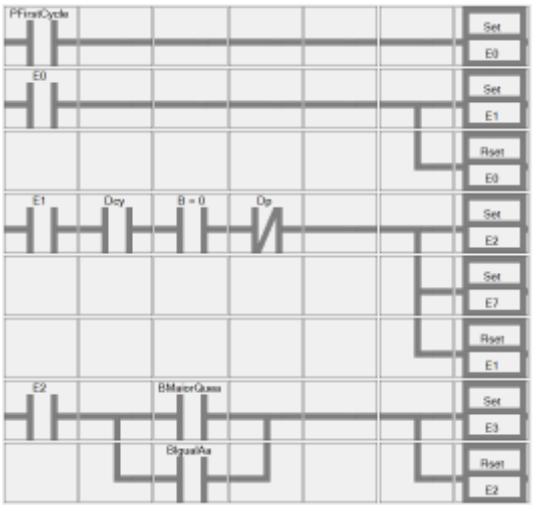


Figura 150 - Primeira parte das funções *Set/ Reset* do sexto exercício

Na Figura 150 observa-se a primeira parte do método das funções de *Set/ Reset*. Os dois primeiros degraus do projeto apenas são executados no primeiro ciclo do projeto, ativando a "etapa 0" e posteriormente a "etapa 1". No terceiro degrau, caso a "etapa 1" esteja ativa, o botão "Dcy" seja pressionado, a balança "B" esteja com o valor zero e o detetor de passagem "DP" não esteja a detetar um briquete, são ativadas a "etapa 2" e a "etapa 7" e desativada a "etapa 1".No último degrau desta parte, quarto degrau, caso a "etapa 2" esteja ativa e o valor da balança "B" seja maior ou igual à quantidade desejada de produto "A", esta etapa deixa de estar ativa passando a estar ativa a "etapa 3".

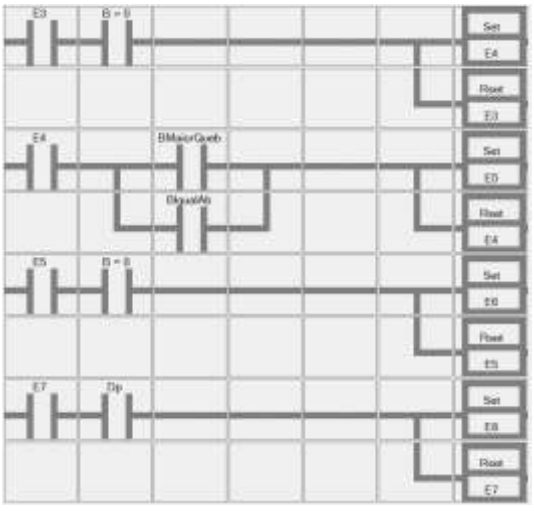


Figura 151 - Segunda parte das funções *Set/ Reset* do sexto exercício

Na Figura 151 observa-se a segunda parte do método das funções de *Set/ Reset*. No primeiro degrau desta parte, quinto degrau, caso a "etapa 3" esteja ativa e o valor da balança "B" seja zero, esta etapa deixa de estar ativa passando a estar ativa a "etapa 4". No sexto degrau, caso a

"etapa 4" esteja ativa e o valor da balança "B" seja maior ou igual à quantidade desejada de produto "B", passa a estar ativa a "etapa 5" e é desativada a "etapa 4". No sétimo degrau, caso a "etapa 5" esteja ativa e o valor da balança "B" seja zero é ativada a "etapa 6" e desativada a "etapa 5". No último degrau desta parte, oitavo degrau, caso a "etapa 7" esteja ativa e o detetor de passagem "DP" esteja a detetar um briquete, o automatismo desativa a "etapa 7" ativando a "etapa 8".

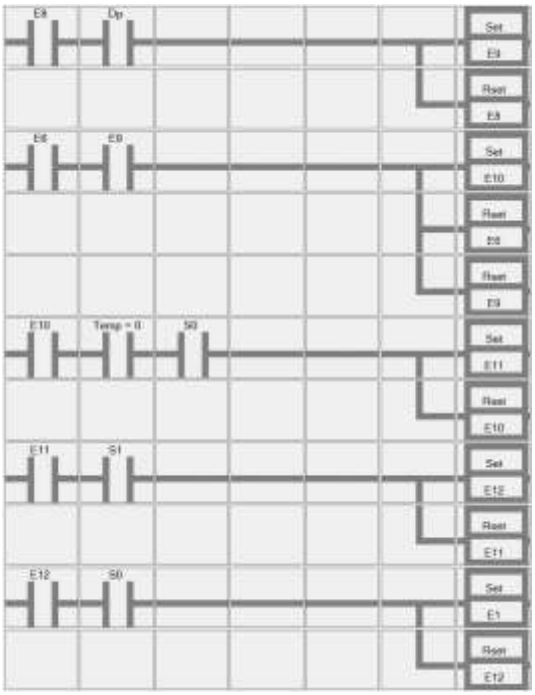


Figura 152 - Terceira parte das funções *Set/ Reset* do sexto exercício

Na Figura 152 observa-se a terceira parte do método das funções de *Set/ Reset*. No primeiro degrau desta parte, nono degrau, caso a "etapa 8" esteja ativa e o detetor de passagem "DP" esteja a detetar um briquete, esta etapa deixa de estar ativa passando a estar ativa a "etapa 9". No décimo degrau, caso a "etapa 6" e a "etapa 9" estejam ativas, passa a estar ativa a "etapa 10" e são desativadas a "etapa 6" e "etapa 9". No décimo-primeiro degrau, caso a "etapa 10" esteja ativa, o temporizador chegue a zero e o fim de curso superior da misturadora "N" esteja pressionado, é ativada a "etapa 1"1 e desativada a "etapa 10". No décimo-segundo degrau, caso a "etapa 11" esteja ativa e o fim de curso inferior da misturadora "N" esteja pressionado, é ativada a "etapa 12" e desativada a "etapa 11". No último degrau desta parte, décimo-terceiro degrau, caso a "etapa 12" esteja ativa e o fim de curso superior da misturadora "N" esteja novamente pressionado, o automatismo desativa a "etapa 12" ativando a "etapa 1" e voltando a repetir o ciclo novamente.

Na Figura 148 encontram-se os últimos degraus que se referem às ativações das saídas nas respetivas etapas. Estes degraus são iguais tanto no método de equações lógicas como no método de funções *Set*/ *Reset* pois, independentemente do método usado, as saídas são ativadas da mesma forma.

Para ser possível programar, o respetivo autómato programável o código em *Ladder* é passado para C, podendo ser encontrado no Apêndice U.

Capítulo 6

6. Conclusão e Trabalho Futuro

Este capítulo inclui as conclusões do projeto realizado bem como sugestões para trabalhos futuros na área científica trabalhada.

6.1 Conclusão

O objetivo central deste projeto prendeu-se com o desenvolvimento de um Autómato Programável de dimensões reduzidas, Micro-Autómato, dispondo de 4 entradas e 4 saídas, e de um Ambiente de Desenvolvimento Multilinguagem: linguagem *Ladder*, linguagem *Grafcet* e linguagem C, que permitisse a programação do respetivo autómato, após a criação do projeto na linguagem que o técnico se sentisse mais à vontade.

Analisando as soluções apresentadas no mercado verificou-se que estas apresentavam dimensões maiores que o desejado, devido a uma maior quantidade de portas disponíveis, custo elevados e, em algumas marcas, à necessidade de comprar licenças do *software* de programação das mesmas.

Verificou-se então, a necessidade de criação de um protótipo funcional: um autómato de dimensões reduzidas, e de um ambiente de desenvolvimento multilinguagem para a programação do mesmo.

Após o desenvolvimento do primeiro protótipo verificou-se que a solução obtida poderia ser de mais fácil integração no ambiente a controlar, retirando a alimentação externa por via de um transformador e, assim, seria possível alcançar dimensões mais reduzidas.

Desenvolveu-se, então, o segundo protótipo que, com dimensões mais reduzidas que o primeiro, resolveu todas as limitações anteriormente encontradas. Ambos os protótipos utilizam

relés de estado solido para acionar dispositivos de grande potência, aceitando apenas corrente alternada.

O ambiente de desenvolvimento multilinguagem foi desenvolvido na linguagem C# com recurso à ferramenta *Visual Studio* 2013, o que se apresentou como uma dificuldade, no início do projeto, devido à falta de conhecimentos e experiencia nesta linguagem.

Este ambiente de desenvolvimento permite, então, a criação de projetos nas três linguagens, guardar e abrir os projetos e a programação do autómato.

A experimentação desenvolvida permitiu melhorar e adaptar, a diversos casos possíveis, o bom funcionamento do ambiente de desenvolvimento. Assim, o ambiente de desenvolvimento aceita diversos casos possíveis nas três linguagens, depurando os erros e, no caso das linguagens de *Ladder* e *Grafcet* passa automaticamente o projeto para a linguagem C para posteriormente programação do Micro-Autómato.

Verificou-se então que, podem ser aplicados projetos para condições distintas das enumeradas nos resultados, mas não sendo totalmente abrangente, pois existe um elevado número de combinações dos símbolos, tanto na linguagem *Grafcet* como *Ladder*.

Por fim, sendo um dos objetivos deste trabalho a necessidade de esta ser uma solução de baixo custo, importa referir que o custo total da solução desenvolvida rondou os 100€ em material, pelo que se pode considerar uma solução de baixo custo, quando comparada com as soluções disponíveis no mercado, tendo sido alcançado largamente este objetivo delineado.

Pode então concluir-se que todos objetivos inicialmente propostos foram alcançados, tendo-se obtido o resultado esperado numa solução desenvolvida e testada.

6.2 Trabalho Futuro

Apesar dos objetivos terem sido atingidos, existem alguns aspetos neste trabalho que permitem acrescentar funcionalidades.

O primeiro aspeto prende-se com a possível necessidade da criação de vários diagramas de *Grafcet* com interação entre si, num mesmo projeto. Pretender-se-ia, então, o acrescento da funcionalidade de um novo diagrama na linguagem *Grafcet* no ambiente de desenvolvimento para que o programador conseguisse controlar simultaneamente dois ou mais diagramas. Por exemplo, num parque de estacionamento, um diagrama que controlasse o sistema de entrada no parque, um diagrama que controlasse o sistema de saída do parque e um diagrama que fizesse a gestão

do número lugares disponíveis recorrendo a variáveis utilizadas nos dois diagramas anteriores, sendo um projeto único de gestão de um parque de estacionamento.

O segundo aspeto prende-se com a possível necessidade da comunicação entre autómatos ou sistemas externos de entradas/saídas, aspeto este já pensado no momento da criação do segundo protótipo em que lhe foi acrescentada uma porta de comunicação. Pretender-se-ia a criação, do *hardware*, para um sistema externo de entradas/saídas que comunicaria, pela porta de comunicação, com o autómato programável, podendo ser expandido o número disponível de portas de entrada/saída. Ainda neste aspeto era pretendido um acrescento da funcionalidade de comunicação, no ambiente de desenvolvimento, com os sistemas externos nas linguagens de *Ladder* e *Grafcet*.

Por último, um terceiro aspeto que se prende com a possível necessidade de ser feita uma simulação completa do sistema, antes da montagem no ambiente a controlar. Aqui pretendia-se o acrescento da funcionalidade de virtualização do autómato programável no ambiente de desenvolvimento multilinguagem. O autómato programável enviaria pela porta série o valor das variáveis utilizadas no projeto e o programador teria a perceção do estado em que o sistema se encontrava, através da observação da virtualização do autómato programável, na respetiva linguagem em que o projeto foi elaborado.

Bibliografia

[1] RS, “CPUs para Autómatas Programables,” [Online]. Available: 1. http://pt.rs-online.com/web/c/automatizacion-y-control-de-procesos/automatas-programables-interfaz-hombre-maquina-y-adquisicion-de-datos/cpus-para-automatas-programables/?sort-by=P_breakPrice1&sort-order=asc&pn=1. [Acedido em Dezembro 2013].

[2] Siemens, *Catalogo - Siemens Simatic ST70-2005.*

[3] RS, “CPUs para Autómatas Programables,” [Online]. Available: http://pt.rs-online.com/web/c/automatizacion-y-control-de-procesos/automatas-programables-interfaz-hombre-maquina-y-adquisicion-de-datos/cpus-para-automatas-programables/?sort-by=P_breakPrice1&sort-order=desc&pn=1. [Acedido em Dezembro 2013].

[4] Omron, *DataSheet - CP series CP1H CPU Unit,* 2013.

[5] RS, “SIMATIC STEP 7 Basic V11,” [Online]. Available: http://pt.rs-online.com/web/p/software-de-programacion-para-automatas-programables/7352907/. [Acedido em Dezembro 2013].

[6] RS, “CX-One V4.x Single-User Licence Only,” [Online]. Available: http://pt.rs-online.com/web/p/accesorios-para-automatas-programables/6685477/?origin=PSF_421317|acc. [Acedido em Dezembro 2013].

[7] I. Martins, “Autómatos Programáveis,” 2013. [Online]. Available: http://pt.scribd.com/doc/201283866/Automatos-Programaveis-Roteiro-Pratico.

[8] Wikipedia, “IEC 61131-3,” [Online]. Available: http://en.wikipedia.org/wiki/IEC_61131-3. [Acedido em Dezembro 2013].

[9] “Imagem Funcition Block Diagram,” [Online]. Available: http://infosys.beckhoff.com/content/1033/tcplccontrol/Images/BILD5_9.gif. [Acedido em Dezembro 2013].

[10] “Imagem Sequential Function Chart,” [Online]. Available: http://www2.cs.unibw.de/tools/DiaGenDiaMeta/img/sfc_t.png. [Acedido em Dezembro 2013].

[11] Wikipédia, “Asea Brown Boveri,” [Online]. Available: http://pt.wikipedia.org/wiki/Asea_Brown_Boveri. [Acedido em Dezembro 2013].

[12] "Imagem Logotipo ABB," [Online]. Available: http://e-move.tv/wp-content/uploads/2012/01/ABB12.jpg. [Acedido em Dezembro 2013].

[13] Wikipédia, "ABB Group," [Online]. Available: http://en.wikipedia.org/wiki/ABB_Group. [Acedido em Dezembro 2013].

[14] ABB, *Catalogo - AC500-eCo: your PLC from ABB,* 2013.

[15] "Imagem AC500 PM554-T," [Online]. Available: http://img-europe.electrocomponents.com/largeimages/R7039998-01.jpg. [Acedido em Dezembro 2013].

[16] "Imagem AC500 PM564-R," [Online]. Available: http://img-europe.electrocomponents.com/largeimages/R6888884-01.jpg. [Acedido em Dezembro 2013].

[17] NovaTech, "IEC 61131-3 Programming Tools For OrionLX," [Online]. Available: http://www.novatechweb.com/news/iec-61131-3-programming-tools-orionlx/. [Acedido em Dezembro 2013].

[18] "Imagem Programming Tools," [Online]. Available: http://novatechweb.com/wp-content/uploads/2011/10/oriolx-tools1.png. [Acedido em Dezembro 2013].

[19] Wikipédia, "Mitsubishi," [Online]. Available: http://pt.wikipedia.org/wiki/Mitsubishi. [Acedido em Dezembro 2013].

[20] "Imagem Logotipo da Mitsubishi Electric," [Online]. Available: http://www.ewon.biz/imf/c/eyJtYXNrIjoibmVvX3JpZ2h0LnBuZyIsIm0iOjE0fQ//images/res_plc_connectivity_mitsubishi_official.jpg. [Acedido em Dezembro 2013].

[21] Wikipédia, "Mitsubishi Electric," [Online]. Available: http://en.wikipedia.org/wiki/Mitsubishi_Electric. [Acedido em Dezembro 2013].

[22] Mitsubishi, *Datasheet - The FX1S Product - A new Family of Micro Controllers,* 2013.

[23] Mitsubishi, *Datasheet - Hardware Manual FX1N Series Programmable Controllers,* 2013.

[24] "Imagem FX1S-10MR-DS," [Online]. Available: http://img-europe.electrocomponents.com/largeimages/CD383015-01.jpg. [Acedido em Dezembro 2013].

[25] "Imagem FX1N24MR-ES/UL," [Online]. Available: http://img-europe.electrocomponents.com/largeimages/C399637-01.jpg. [Acedido em Dezembro 2013].

[26] Mitsubishi, *Catalogo Programmable Controller Engineering Software GX Works2,* 2013.

[27] "Imagem PLC Software GX Works 2 FX," [Online]. Available: http://plccompare.com/wp-content/uploads/2011/07/GX-Works2.png. [Acedido em Dezembro 2013].

[28] Wikipédia, "Omron," [Online]. Available: http://pt.wikipedia.org/wiki/Omron. [Acedido em Dezembro 2013].

[29] "Imagem Logotipo da Omron," [Online]. Available: http://www.mpassociates.com/ebooth/images/219812OmronElecCompLogo-%20high%20res.jpg. [Acedido em Dezembro 2013].

[30] Omron, *Datasheet - SYSMAC CP-series CP1E CPU Units,* 2013.

[31] "Imagem CP1E-E," [Online]. Available: http://img-europe.electrocomponents.com/largeimages/R7055795-01.jpg. [Acedido em Dezembro 2013].

[32] "Imagem CP1H CPU 40," [Online]. Available: http://img-europe.electrocomponents.com/largeimages/R7462890-01.jpg. [Acedido em Dezembro 2013].

[33] Omron, *Manual - CX-Programmer Operation Manual,* 2013.

[34] "Imagem Logotipo Siemens," [Online]. Available: http://www.power-eng.com/content/pe/en/articles/2013/11/siemens-responds-to-us-power-market-uncertainty/_jcr_content/leftcolumn/article/thumbnailimage.img.jpg. [Acedido em Dezembro 2013].

[35] Wikipédia, "Siemens_AG," [Online]. Available: http://pt.wikipedia.org/wiki/Siemens_AG. [Acedido em Dezembro 2013].

[36] Siemens, *Datasheet - Siemens Product data sheet 6ES7211-1HE31-0XB0,* 2013.

[37] Siemens, *Datasheet - Siemens Simatic S7-300,* 2013.

[38] "Imagem Siemens CPU1211C," [Online]. Available: http://img-europe.electrocomponents.com/largeimages/R6683042-20.jpg. [Acedido em Dezembro 2013].

[39] "Imagem Siemens CPU314C-2PN/DP," [Online]. Available: http://img-europe.electrocomponents.com/largeimages/R432715-01.jpg. [Acedido em Dezembro 2013].

[40] "Imagem Simatic Step 7," [Online]. Available: http://www.automation.siemens.com/mcms/simatic-controller-software/en/step7/step7-professional/PublishingImages/SIMATIC_STEP7_Basic_software.jpg. [Acedido em Dezembro 2013].

[41] Siemens, *Manual - Siemens - Products for Totally Integrated Automatio,* 2013.

[42] J. A. F. Ferreira, *Tese de Mestrado - Virtualização de autómatos programáveis,* Universidade de Aveiro, 1994.

[43] R.P.A. van Haendel, "Design of an omnidirectional universal mobile platform", DCT 2005.117, DCT traineeship report, Eindhoven, September 2005.

[44] Daniel W. Hart, "Power Electronics", ISBN 978-0-07-338067-4, MHID 0-07-338067-9, Valparaiso University, Valparaiso, Indiana, 2011.

[45] Ned Mohan, Tore M. Undeland, William P. Robbins, "Power Electronics: Converters, Applications and Design", Second Edition, ISBN 0-471-58408-8, University of Minnesota, Minneapolis, Minnesota, EUA, 1995.

[46] Jorge Villagra, David Herrero-Pérez, "A Comparison of Control Techniques for Robust Docking Maneuvers of an AGV", IEEE Transactions on Control Systems Technology, Vol. 20, No. 4, July 2012.

[47] Florentina Adăscăliţei, Ioan Doroftei, "Practical Applications for Mobile Robots based on Mecanum Wheels - a Systematic Survey", Gh. Asachi Technical University of Iasi, Mechanical Engineering Faculty, Theory of Mechanisms and Robotics Department, B-dul D. Mangeron, 61-63, 700050, Iasi, Romania, MECAHITECH'11, Vol. 3, 2011.

[48] Christina Tsalicoglou, Xavier Perrin, "Survey on Navigation Assistants for People with Disabilities", Autonomous Systems Laboratory, ETHZ, Zurich, Switzerland, 2010.

[49] Desmond King-Hele, "Erasmus Darwin's Improved Design for Steering Carriages – and Cars", The Royal Society, London, 2002.

[50] Marcy Lowe, Saori Tokuoka, Tali Trigg, Gary Gereffi, "Lithium-ion Batteries for Electric Vehicles: The U.S. Value Chain", Center on Globalization, Governance & Competitiveness, Duke University, Durham, North Carolina, October 5, 2010.

[51] B. G. Kim, F. P. Tredeau, Z. M. Salameh, "Fast Chargeability Lithium Polymer Batteries", Department of Electrical and Computer Engineering, University of Massachusetts Lowell, Lowell, MA 01854, 2008.

[52] Onib Nasir, Musman Yousuf, "Introducing: The Mecanum Wheel", IEEE PNEC PERSPECTIVE, Vol. 4, Autumn 2012.

[53] John R. Miller, Patrice Simon, "Fundamentals of Electrochemical Capacitor Design and Operation", The Electrochemical Society, Spring 2008.

[54] Ryan Thomas, "Omni-Directional Mobile Platform for The Transportation of Heavy Objects", Massey University, Palmerston North, New Zealand, 2011.

[55] Mohamed Hedi Chabchoub, Hafedh Trabelsi, "Consolidation of the Electric Vehicle Battery by an Ultracapacitor for Performance Improvement", Computer Embedded System (CES), National School of Engineers of Sfax, 10th International Multi-Conference on Systems, Signals & Devices (SSD), Hammamet, Tunisia, March 18-21, 2013.

[56] Melvin M. Morrison, "Inertial Measurement Unit", US Patent 4711125 A, December 8, 1987.

[57] Anders Hejlsberg, Mads Torgersen, Scott Wiltamuth, Peter Golde, "The C# Programming Language", Fourth Edition, ISBN 978-0-321-74176-9, Microsoft Corporation, Redmond, King County, Washington, USA, 2010.

Apêndices

Apêndice A – Funções_Usart.c

```
#include "Funcoes_USART.h"

void InicializarUSART(void)
{
	#define BAUD USART_BAUDRATE
	#include <util/setbaud.h>
	UBRR1H = UBRRH_VALUE;
	UBRR1L = UBRRL_VALUE;
	#if USE_2X
	  UCSR1A |= _BV(U2X1);
	#else
	  UCSR1A &= ~_BV(U2X1);
	#endif
	#undef BAUD
	UCSR1B = (_BV(RXCIE1) | _BV(TXCIE1) | _BV(RXEN1) | _BV(TXEN1)) & (~_BV(UCSZ12) & ~_BV(UDRIE1) &
~_BV(TXB81) & ~_BV(RXB81));
	UCSR1C = (_BV(UCSZ11) | _BV(UCSZ10)) & (~_BV(UMSEL11) & ~_BV(UMSEL10) & ~_BV(UPM11) & ~_BV(UPM10)
& ~_BV(USBS1) & ~_BV(UCPOL1));
}

void PermitirInterruptUSART(void)
{
	UCSR1A |= _BV(RXC1) | _BV(TXC1);
	UCSR1B |= _BV(RXCIE1) | _BV(TXCIE1);
}

void ImpedirInterruptUSART(void)
{
	UCSR1A |= _BV(RXC1) | _BV(TXC1);
	UCSR1B &= ~_BV(RXCIE1) & ~_BV(TXCIE1);
}

void EnviarByteUSART(uint8_t Byte)
{
	do {} while(!(UCSR1A & _BV(UDRE1)));
	UDR1 = Byte;
}

uint8_t LerByteUSART(void)
{
	do {} while (!(UCSR1A & _BV(RXC1)));
	return UDR1;
}
```

Apêndice B – Funções_Usart.h

```
#ifndef FUNCOES_USART_H_
#define FUNCOES_USART_H_

#define F_CPU 16000000UL
#define USART_BAUDRATE 57600

#include <avr/io.h>

void InicializarUSART(void);
void PermitirInterruptUSART(void);
void ImpedirInterruptUSART(void);
void EnviarByteUSART(uint8_t Byte);
uint8_t LerByteUSART(void);

#endif
```

Apêndice C – Funções_ADC.c

```
#include "Funcoes_ADC.h"

void InicializarADC(void)
{
	ADCSRA = (_BV(ADEN) | _BV(ADIF) | _BV(ADPS1)) & (~_BV(ADSC) & ~_BV(ADATE) & ~_BV(ADIE) & ~_BV(ADPS2) &
~_BV(ADPS0));
	ADCSRB = 0x00;
	ADMUX = 0x00;
	DIDR0 = 0x00;
}

uint8_t LerValor8bitADC(uint8_t canal)
{
	uint8_t Valor_ADC = 0;
	ADMUX = (_BV(REFS0) | _BV(ADLAR)) & (~_BV(REFS1) & ~_BV(MUX3) & ~_BV(MUX2) & ~_BV(MUX1) &
~_BV(MUX0));
	ADMUX |= canal;
	DIDR0 |= (1 << canal);
	ADCSRA = (_BV(ADEN) | _BV(ADSC) | _BV(ADIF) | _BV(ADPS1)) & (~_BV(ADPS2) & ~_BV(ADPS0) & ~_BV(ADATE) &
~_BV(ADIE));
	do{} while(!(ADCSRA & _BV(ADIF)));
	Valor_ADC = ADCH;
	DIDR0 = 0x00;
	return Valor_ADC;
}
```

Apêndice D – Funções_ADC.h

```
#ifndef FUNCOES_ADC_H_
#define FUNCOES_ADC_H_

#include <avr/io.h>

void InicializarADC(void);
uint8_t LerValor8bitADC(uint8_t canal);

#endif
```

Apêndice E – Código em C do código em Grafcet do 1º Exercício

```
#include "Funcoes_USART.h"
#include "Funcoes_ADC.h"
#include <avr/io.h>
#include <avr/interrupt.h>

int main (void)
{
	DDRD |= _BV(DDD5);
	PORTD &= ~_BV(PORTD5);
	DDRD |= _BV(DDD4);
	PORTD &= ~_BV(PORTD4);
	DDRD |= _BV(DDD3);
	PORTD &= ~_BV(PORTD3);
	DDRD |= _BV(DDD2);
	PORTD &= ~_BV(PORTD2);
	uint8_t Etapa0 = 1;
	uint8_t Etapa1 = 0;
	uint8_t Etapa2 = 0;
	uint8_t Etapa3 = 0;
	uint8_t S1 = 0;
	uint8_t S2 = 0;
	uint8_t S0 = 0;
	InicializarUSART();
	ImpedirInterruptUSART();
	InicializarADC();
	sei();
	while(1)
	{
		S1 = (LerValor8bitADC(2)*0.4) + (S1*0.6);
		S2 = (LerValor8bitADC(3)*0.4) + (S2*0.6);
		S0 = (LerValor8bitADC(4)*0.4) + (S0*0.6);
		if(Etapa0 == 1)
		{
			Etapa1 = 1;
			Etapa0 = 0;
		}
		if(Etapa1 == 1)
		{
			PORTD &= ~_BV(PORTD5);
			PORTD &= ~_BV(PORTD4);
			PORTD &= ~_BV(PORTD3);
			PORTD &= ~_BV(PORTD2);
		}
		if(((S1>220)&&(S0<50)) && Etapa1 == 1)
		{
			Etapa2 = 1;
			Etapa1 = 0;
		}
		if(Etapa2 == 1)
		{
			PORTD |= _BV(PORTD5);
			PORTD &= ~_BV(PORTD4);
			PORTD |= _BV(PORTD3);
			PORTD &= ~_BV(PORTD2);
		}
		if(((S0>220)||(S2>220)) && Etapa2 == 1)
		{
```

```
                        Etapa1 = 1;
                        Etapa2 = 0;
                }
                if(((S2>220)&&(S0<50)) && Etapa1 == 1)
                {
                        Etapa3 = 1;
                        Etapa1 = 0;
                }
                if(Etapa3 == 1)
                {
                        PORTD &= ~_BV(PORTD5);
                        PORTD |= _BV(PORTD4);
                        PORTD &= ~_BV(PORTD3);
                        PORTD |= _BV(PORTD2);
                }
                if(((S0>220)||(S1>220)) && Etapa3 == 1)
                {
                        Etapa1 = 1;
                        Etapa3 = 0;
                }
        }
}
```

Apêndice F – Código em C do código Ladder (método de equações lógicas) do 1º Exercício

```
#include "Funcoes_USART.h"
#include "Funcoes_ADC.h"
#include <avr/io.h>
#include <avr/interrupt.h>

uint8_t PFirstCycle = 1;
uint8_t S0 = 0;
uint8_t S1 = 0;
uint8_t S2 = 0;
uint8_t E0 = 0;
uint8_t E1 = 0;
uint8_t E2 = 0;
uint8_t E3 = 0;

int main (void)
{
	DDRD |= _BV(DDD7);
	PORTD &= ~_BV(PORTD7);
	DDRD |= _BV(DDD6);
	PORTD &= ~_BV(PORTD6);
	DDRD |= _BV(DDD5);
	PORTD &= ~_BV(PORTD5);
	DDRD |= _BV(DDD4);
	PORTD &= ~_BV(PORTD4);
	InicializarUSART();
	ImpedirInterruptUSART();
	InicializarADC();
	sei();
	while(1)
	{
		S0 = (LerValor8bitADC(2)*0.4) + (S0*0.6);
		S1 = (LerValor8bitADC(3)*0.4) + (S1*0.6);
		S2 = (LerValor8bitADC(4)*0.4) + (S2*0.6);
		if((PFirstCycle == 1))
		{
			E0 = 1;
		}
		if(((E0 == 1) || ((E3 == 1) && (S1 == 255)) || ((E3 == 1) && (S0 == 255)) || ((E2 == 1) && (S2 == 255))
|| ((E2 == 1) && (S0 == 255)) || ((E1 == 1))) && (E2 == 0) && (E3 == 0))
		{
			E1 = 1;
		}
		if(((E2 == 1) || ((E1 == 1) && (S1 == 255) && (S0 == 0))) && (E3 == 0))
		{
			E2 = 1;
		}
		if(((E3 == 1) || ((E1 == 1) && (S2 == 255) && (S0 == 0))) && (E2 == 0))
		{
			E3 = 1;
		}
		if((E2 == 1))
		{
			PORTD |= _BV(PORTD4);
			PORTD |= _BV(PORTD6);
			PORTD &= ~_BV(PORTD5);
```

```
                PORTD &= ~_BV(PORTD3);
            }
            if((E3 == 1))
            {
                PORTD |= _BV(PORTD3);
                PORTD |= _BV(PORTD5);
                PORTD &= ~_BV(PORTD6);
                PORTD &= ~_BV(PORTD4);
            }
            if((E1 == 1))
            {
                E0 = 0;
            }
            if(((E2 == 1) || ((E3 == 1))))
            {
                E1 = 0;
            }
            if(((E1 == 1) || ((E3 == 1))))
            {
                E2 = 0;
            }
            if(((E2 == 1) || ((E1 == 1))))
            {
                E3 = 0;
            }
        }
}
```

Apêndice G – Código em C do código Ladder (método de funções *Set/ Reset*) do 1º Exercício

```
#include "Funcoes_USART.h"
#include "Funcoes_ADC.h"
#include <avr/io.h>
#include <avr/interrupt.h>

uint8_t PFirstCycle = 1;
uint8_t S0 = 0;
uint8_t S1 = 0;
uint8_t S2 = 0;
uint8_t E0 = 0;
uint8_t E1 = 0;
uint8_t E2 = 0;
uint8_t E3 = 0;

int main (void)
{
	DDRD |= _BV(DDD7);
	PORTD &= ~_BV(PORTD7);
	DDRD |= _BV(DDD6);
	PORTD &= ~_BV(PORTD6);
	DDRD |= _BV(DDD5);
	PORTD &= ~_BV(PORTD5);
	DDRD |= _BV(DDD4);
	PORTD &= ~_BV(PORTD4);
	InicializarUSART();
	ImpedirInterruptUSART();
	InicializarADC();
	sei();
	while(1)
	{
		S0 = (LerValor8bitADC(2)*0.4) + (S0*0.6);
		S1 = (LerValor8bitADC(3)*0.4) + (S1*0.6);
		S2 = (LerValor8bitADC(4)*0.4) + (S2*0.6);
		if((PFirstCycle == 1))
		{
			E0 = 1;
		}
		if((E0 == 1))
		{
			E0 = 0;
			E1 = 1;
		}
		if((E1 == 1) && (S1 == 255) && (S0 == 0))
		{
			E1 = 0;
			E2 = 1;
		}
		if((E1 == 1) && (S2 == 255) && (S0 == 0))
		{
			E1 = 1;
			E3 = 1;
		}
		if(((E2 == 1) && ((S2 == 255) || (S0 == 255))))
		{
			E2 = 0;
```

```
                        E1 = 1;
                }
                if(((E3 == 1) && ((S1 == 255) || (S0 == 255))))
                {
                        E3 = 0;
                        E1 = 1;
                }
                if((E2 == 1))
                {
                        PORTD |= _BV(PORTD4);
                        PORTD |= _BV(PORTD6);
                        PORTD &= ~_BV(PORTD5);
                        PORTD &= ~_BV(PORTD3);
                }
                if((E3 == 1))
                {
                        PORTD |= _BV(PORTD3);
                        PORTD |= _BV(PORTD5);
                        PORTD &= ~_BV(PORTD6);
                        PORTD &= ~_BV(PORTD4);
                }
        }
}
```

Apêndice H – Código em C do código em Grafcet do 2º Exercício

```
#include "Funcoes_USART.h"
#include "Funcoes_ADC.h"
#include <avr/io.h>
#include <avr/interrupt.h>

static void Config_Timer1_Interrupt_1Hz(void);
int main (void)
{
	DDRD |= _BV(DDD5);
	PORTD &= ~_BV(PORTD5);
	DDRD |= _BV(DDD4);
	PORTD &= ~_BV(PORTD4);
	DDRD |= _BV(DDD3);
	PORTD &= ~_BV(PORTD3);
	DDRD |= _BV(DDD2);
	PORTD &= ~_BV(PORTD2);
	uint8_t Etapa0 = 1;
	uint8_t Etapa1 = 0;
	uint8_t Etapa2 = 0;
	uint8_t Etapa3 = 0;
	uint8_t Etapa4 = 0;
	uint8_t Etapa5 = 0;
	uint8_t Rx = 0;
	uint8_t Fc1 = 0;
	uint8_t Fc2 = 0;
	uint8_t Ir = 0;
	InicializarUSART();
	ImpedirInterruptUSART();
	InicializarADC();
	Config_Timer1_Interrupt_1Hz();
	sei();
	while(1)
	{
		Rx = (LerValor8bitADC(2)*0.4) + (Rx*0.6);
		Fc1 = (LerValor8bitADC(3)*0.4) + (Fc1*0.6);
		Fc2 = (LerValor8bitADC(4)*0.4) + (Fc2*0.6);
		Ir = (LerValor8bitADC(5)*0.4) + (Ir*0.6);
		if(((Rx>220)) && Etapa0 == 1)
		{
			Etapa1 = 1;
			Etapa0 = 0;
		}
		if(Etapa1 == 1)
		{
			PORTD |= _BV(PORTD5);
			PORTD |= _BV(PORTD4);
			PORTD |= _BV(PORTD3);
			PORTD &= ~_BV(PORTD2);
		}
		if(((Fc1>220)||(Fc2>220)) && Etapa1 == 1)
		{
			Etapa5 = 1;
			Etapa1 = 0;
		}
		if(Etapa5 == 1)
		{
			PORTD &= ~_BV(PORTD5);
			PORTD &= ~_BV(PORTD4);
```

```
                        PORTD &= ~_BV(PORTD3);
                        PORTD &= ~_BV(PORTD2);
                        Temp=0;
                }
                if(((Fc1>220)&&(Fc2>220)) && Etapa5 == 1)
                {
                        Etapa3 = 1;
                        Etapa5 = 0;
                }
                if(Etapa3 == 1)
                {
                        PORTD &= ~_BV(PORTD5);
                        PORTD &= ~_BV(PORTD4);
                        PORTD &= ~_BV(PORTD3);
                        PORTD &= ~_BV(PORTD2);
                }
                if(((Temp==60)&&(Ir<50)) && Etapa3 == 1)
                {
                        Etapa4 = 1;
                        Etapa3 = 0;
                }
                if(Etapa4 == 1)
                {
                        PORTD &= ~_BV(PORTD5);
                        PORTD |= _BV(PORTD4);
                        PORTD |= _BV(PORTD3);
                        PORTD &= ~_BV(PORTD2);
                }
                if(((Rx>220)||(Ir>220)) && Etapa4 == 1)
                {
                        Etapa1 = 1;
                        Etapa4 = 0;
                }
                if(((Temp==40)) && ((Rx>220)) && Etapa4 == 1)
                {
                        Etapa1 = 1;
                        Etapa4 = 0;
                }
        }
}
ISR(TIMER1_COMPA_vect)
{
        if(Etapa3 == 1)
        {
          Temp += 1;
        }
        if(Etapa4 == 1)
        {
          Temp -= 1;
        }
}
static void Config_Timer1_Interrupt_1Hz(void)
{
        TCCR1A = 0x00;
        TCCR1C = 0x00;
        TCNT1 = 0x00;
        OCR1A = 15624;
        TIFR1 |= _BV(OCF1A);
        TIMSK1 |= _BV(OCIE1A);
        TCCR1B = (_BV(WGM12) | _BV(CS12) | _BV(CS10)) & (~_BV(WGM13) & ~_BV(CS11) & ~_BV(ICNC1) &
~_BV(ICES1));
}
```

Apêndice I – Código em C do código Ladder (método de equações lógicas) do 2º Exercício

```
#include "Funcoes_USART.h"
#include "Funcoes_ADC.h"
#include <avr/io.h>
#include <avr/interrupt.h>

static void Config_Timer1_Interrupt_1Hz(void);

uint8_t PFirstCycle = 1;
uint8_t Rx = 0;
uint8_t Fc1 = 0;
uint8_t Fc2 = 0;
uint8_t Ir = 0;
uint8_t Temp = 0;
uint8_t aux_Temp = 0;
uint8_t E0 = 0;
uint8_t E1 = 0;
uint8_t E2 = 0;
uint8_t E3 = 0;
uint8_t E4 = 0;
uint8_t Temp2 = 0;
uint8_t aux_Temp2 = 0;

int main (void)
{
	DDRD |= _BV(DDD7);
	PORTD &= ~_BV(PORTD7);
	DDRD |= _BV(DDD6);
	PORTD &= ~_BV(PORTD6);
	DDRD |= _BV(DDD5);
	PORTD &= ~_BV(PORTD5);
	DDRD |= _BV(DDD4);
	PORTD &= ~_BV(PORTD4);
	InicializarUSART();
	ImpedirInterruptUSART();
	InicializarADC();
	Config_Timer1_Interrupt_1Hz();
	sei();
	while(1)
	{
		if(E1 == 1)
		{
			aux_Temp = 0;
			aux_Temp2 = 0;
		}
		Rx = (LerValor8bitADC(2)*0.4) + (Rx*0.6);
		Fc1 = (LerValor8bitADC(3)*0.4) + (Fc1*0.6);
		Fc2 = (LerValor8bitADC(4)*0.4) + (Fc2*0.6);
		Ir = (LerValor8bitADC(5)*0.4) + (Ir*0.6);
		if((PFirstCycle == 1))
		{
			E0 = 1;
		}
		if(((E1 == 1) || ((E4 == 1) && (Temp2 == 0) && (Rx == 255)) || ((E4 == 1) && (Ir == 255)) || ((E4 == 1)
&& (Rx == 255)) || ((E0 == 1) && (Rx == 255))) && (E2 == 0))
		{
```

```
                E1 = 1;
        }
        if(((E2 == 1) || ((E1 == 1) && (Fc2 == 255)) || ((E1 == 1) && (Fc1 == 255))) && (E3 == 0))
        {
                E2 = 1;
        }
        if(((E3 == 1) || ((E2 == 1) && (Fc1 == 255) && (Fc2 == 255))) && (E4 == 0))
        {
                E3 = 1;
        }
        if(((E4 == 1) || ((E3 == 1) && (Temp == 0) && (Ir == 0))))
        {
                E4 = 1;
        }
        if((E1 == 1))
        {
                PORTD |= _BV(PORTD6);
                PORTD |= _BV(PORTD4);
                PORTD |= _BV(PORTD5);
                PORTD &= ~_BV(PORTD3);
        }
        if((E3 == 1))
        {
                if(aux_Temp == 0)
                {
                        Temp = 60;
                        aux_Temp = 1;
                }
        }
        if((E4 == 1))
        {
                if(aux_Temp2 == 1)
                {
                        Temp2 = 20;
                        aux_Temp2 = 2;
                }
                PORTD |= _BV(PORTD4);
                PORTD |= _BV(PORTD5);
                PORTD &= ~_BV(PORTD6);
                PORTD &= ~_BV(PORTD3);
        }
        if((E1 == 1))
        {
                E0 = 0;
        }
        if((E2 == 1))
        {
                E1 = 0;
        }
        if((E3 == 1))
        {
                E2 = 0;
        }
        if((E4 == 1))
        {
                E3 = 0;
        }
        if((E1 == 1))
        {
                E4 = 0;
        }
}
```

```
}

ISR(TIMER1_COMPA_vect)
{
	if(E3 == 1)
	{
		Temp -= 1;
	}
	if(E4 == 1)
	{
		Temp2 -= 1;
	}
}

static void Config_Timer1_Interrupt_1Hz(void)
{
	TCCR1A = 0x00;
	TCCR1C = 0x00;
	TCNT1 = 0x00;
	OCR1A = 15624;
	TIFR1 |= _BV(OCF1A);
	TIMSK1 |= _BV(OCIE1A);
	TCCR1B = (_BV(WGM12) | _BV(CS12) | _BV(CS10)) & (~_BV(WGM13) & ~_BV(CS11) & ~_BV(ICNC1) &
~_BV(ICES1));
}
```

Apêndice J – Código em C do código Ladder (método de funções *Set/ Reset*) do 2º Exercício

```
#include "Funcoes_USART.h"
#include "Funcoes_ADC.h"
#include <avr/io.h>
#include <avr/interrupt.h>

static void Config_Timer1_Interrupt_1Hz(void);

uint8_t PFirstCycle = 1;
uint8_t Rx = 0;
uint8_t Fc1 = 0;
uint8_t Fc2 = 0;
uint8_t Ir = 0;
uint8_t Temp = 0;
uint8_t aux_Temp = 0;
uint8_t E0 = 0;
uint8_t E1 = 0;
uint8_t E2 = 0;
uint8_t E3 = 0;
uint8_t E4 = 0;
uint8_t Temp2 = 0;
uint8_t aux_Temp2 = 0;

int main (void)
{
	DDRD |= _BV(DDD7);
	PORTD &= ~_BV(PORTD7);
	DDRD |= _BV(DDD6);
	PORTD &= ~_BV(PORTD6);
	DDRD |= _BV(DDD5);
	PORTD &= ~_BV(PORTD5);
	DDRD |= _BV(DDD4);
	PORTD &= ~_BV(PORTD4);
	InicializarUSART();
	ImpedirInterruptUSART();
	InicializarADC();
	Config_Timer1_Interrupt_1Hz();
	sei();
	while(1)
	{
		if(E1 == 1)
		{
			aux_Temp = 0;
			aux_Temp2 = 0;
		}
		Rx = (LerValor8bitADC(2)*0.4) + (Rx*0.6);
		Fc1 = (LerValor8bitADC(3)*0.4) + (Fc1*0.6);
		Fc2 = (LerValor8bitADC(4)*0.4) + (Fc2*0.6);
		Ir = (LerValor8bitADC(5)*0.4) + (Ir*0.6);
		if((PFirstCycle == 1))
		{
			E0 = 1;
		}
		if((E0 == 1) && (Rx == 255))
		{
			E0 = 0;
```

```
                E1 = 1;
        }
        if(((E1 == 1) && ((Fc2 == 255) || (Fc1 == 255))))
        {
                E1 = 0;
                E2 = 1;
        }
        if((E2 == 1) && (Fc1 == 255) && (Fc2 == 255))
        {
                E2 = 0;
                E3 = 1;
        }
        if((E3 == 1) && (Temp == 0) && (Ir == 0))
        {
                E3 = 0;
                E4 = 1;
        }
        if(((E4 == 1) && (((Temp2 == 0) && (Rx == 255)) || (Ir == 255) || (Rx == 255))))
        {
                E4 = 0;
                E1 = 1;
        }
        if((E1 == 1))
        {
                PORTD |= _BV(PORTD6);
                PORTD |= _BV(PORTD4);
                PORTD |= _BV(PORTD5);
                PORTD &= ~_BV(PORTD3);
        }
        if((E3 == 1))
        {
                if(aux_Temp == 0)
                {
                        Temp = 60;
                        aux_Temp = 1;
                }
        }
        if((E4 == 1))
        {
                if(aux_Temp2 == 1)
                {
                        Temp2 = 20;
                        aux_Temp2 = 2;
                }
                PORTD |= _BV(PORTD4);
                PORTD |= _BV(PORTD5);
                PORTD &= ~_BV(PORTD6);
                PORTD &= ~_BV(PORTD3);
        }
    }
}

ISR(TIMER1_COMPA_vect)
{
    if(E3 == 1)
    {
        Temp -= 1;
    }
    if(E4 == 1)
    {
        Temp2 -= 1;
    }
```

```
}

static void Config_Timer1_Interrupt_1Hz(void)
{
        TCCR1A = 0x00;
        TCCR1C = 0x00;
        TCNT1 = 0x00;
        OCR1A = 15624;
        TIFR1 |= _BV(OCF1A);
        TIMSK1 |= _BV(OCIE1A);
        TCCR1B = (_BV(WGM12) | _BV(CS12) | _BV(CS10)) & (~_BV(WGM13) & ~_BV(CS11) & ~_BV(ICNC1) &
~_BV(ICES1));
}
```

Apêndice K – Código em C do código em Grafcet do 3º Exercício

```
#include "Funcoes_USART.h"
#include "Funcoes_ADC.h"
#include <avr/io.h>
#include <avr/interrupt.h>

static void Config_Timer1_Interrupt_1Hz(void);

int main (void)
{
	DDRD |= _BV(DDD5);
	PORTD &= ~_BV(PORTD5);
	DDRD |= _BV(DDD4);
	PORTD &= ~_BV(PORTD4);
	DDRD |= _BV(DDD3);
	PORTD &= ~_BV(PORTD3);
	DDRD |= _BV(DDD2);
	PORTD &= ~_BV(PORTD2);
	DDRD |= _BV(DDD1);
	PORTD &= ~_BV(PORTD1);
	DDRD |= _BV(DDD0);
	PORTD &= ~_BV(PORTD0);
	uint8_t Etapa0 = 1;
	uint8_t Etapa1 = 0;
	uint8_t Etapa2 = 0;
	uint8_t Etapa3 = 0;
	uint8_t Etapa4 = 0;
	uint8_t Etapa5 = 0;
	uint8_t Etapa6 = 0;
	uint8_t Etapa7 = 0;
	uint8_t Etapa8 = 0;
	uint8_t Etapa9 = 0;
	uint8_t IL = 0;
	InicializarUSART();
	ImpedirInterruptUSART();
	InicializarADC();
	Config_Timer1_Interrupt_1Hz();
	sei();
	while(1)
	{
		IL = (LerValor8bitADC(2)*0.4) + (IL*0.6);
		if(Etapa0 == 1)
		{
			Etapa1 = 1;
			Etapa0 = 0;
		}
		if(Etapa1 == 1)
		{
			PORTD &= ~_BV(PORTD5);
			PORTD &= ~_BV(PORTD4);
			PORTD &= ~_BV(PORTD3);
			PORTD &= ~_BV(PORTD2);
			PORTD &= ~_BV(PORTD1);
			PORTD &= ~_BV(PORTD0);
			Timer=70;
		}
		if(((IL<50)) && Etapa1 == 1)
		{
```

```
		Etapa2 = 1;
		Etapa1 = 0;
}
if(Etapa2 == 1)
{
		PORTD &= ~_BV(PORTD5);
		PORTD &= ~_BV(PORTD4);
		PORTD &= ~_BV(PORTD3);
		PORTD &= ~_BV(PORTD2);
		PORTD |= _BV(PORTD1);
		PORTD |= _BV(PORTD0);
}
if(((Timer==67)) && Etapa2 == 1)
{
		Etapa3 = 1;
		Etapa2 = 0;
}
if(Etapa3 == 1)
{
		PORTD &= ~_BV(PORTD5);
		PORTD &= ~_BV(PORTD4);
		PORTD &= ~_BV(PORTD3);
		PORTD &= ~_BV(PORTD2);
		PORTD &= ~_BV(PORTD1);
		PORTD &= ~_BV(PORTD0);
}
if(((Timer==65)) && Etapa3 == 1)
{
		Etapa1 = 1;
		Etapa3 = 0;
}
if(((IL>220)) && Etapa1 == 1)
{
		Etapa4 = 1;
		Etapa1 = 0;
}
if(Etapa4 == 1)
{
		PORTD |= _BV(PORTD5);
		PORTD &= ~_BV(PORTD4);
		PORTD &= ~_BV(PORTD3);
		PORTD |= _BV(PORTD2);
		PORTD &= ~_BV(PORTD1);
		PORTD &= ~_BV(PORTD0);
}
if(((Timer==40)) && Etapa4 == 1)
{
		Etapa5 = 1;
		Etapa4 = 0;
}
if(Etapa5 == 1)
{
		PORTD |= _BV(PORTD5);
		PORTD &= ~_BV(PORTD4);
		PORTD &= ~_BV(PORTD3);
		PORTD &= ~_BV(PORTD2);
		PORTD &= ~_BV(PORTD1);
		PORTD |= _BV(PORTD0);
}
if(((Timer==37)) && Etapa5 == 1)
{
		Etapa6 = 1;
```

```
                Etapa5 = 0;
        }
        if(Etapa6 == 1)
        {
                PORTD |= _BV(PORTD5);
                PORTD |= _BV(PORTD4);
                PORTD &= ~_BV(PORTD3);
                PORTD &= ~_BV(PORTD2);
                PORTD &= ~_BV(PORTD1);
                PORTD &= ~_BV(PORTD0);
        }
        if(((Timer==35)) && Etapa6 == 1)
        {
                Etapa7 = 1;
                Etapa6 = 0;
        }
        if(Etapa7 == 1)
        {
                PORTD &= ~_BV(PORTD5);
                PORTD |= _BV(PORTD4);
                PORTD |= _BV(PORTD3);
                PORTD &= ~_BV(PORTD2);
                PORTD &= ~_BV(PORTD1);
                PORTD &= ~_BV(PORTD0);
        }
        if(((Timer==5)) && Etapa7 == 1)
        {
                Etapa8 = 1;
                Etapa7 = 0;
        }
        if(Etapa8 == 1)
        {
                PORTD &= ~_BV(PORTD5);
                PORTD |= _BV(PORTD4);
                PORTD &= ~_BV(PORTD3);
                PORTD &= ~_BV(PORTD2);
                PORTD |= _BV(PORTD1);
                PORTD &= ~_BV(PORTD0);
        }
        if(((Timer==2)) && Etapa8 == 1)
        {
                Etapa9 = 1;
                Etapa8 = 0;
        }
        if(Etapa9 == 1)
        {
                PORTD |= _BV(PORTD5);
                PORTD |= _BV(PORTD4);
                PORTD &= ~_BV(PORTD3);
                PORTD &= ~_BV(PORTD2);
                PORTD &= ~_BV(PORTD1);
                PORTD &= ~_BV(PORTD0);
        }
        if(((Timer=0)) && Etapa9 == 1)
        {
                Etapa1 = 1;
                Etapa9 = 0;
        }
    }
}

ISR(TIMER1_COMPA_vect)
```

```
{
	if(Etapa2 == 1)
	{
		Timer -= 1;
	}
	if(Etapa3 == 1)
	{
		Timer -= 1;
	}
	if(Etapa4 == 1)
	{
		Timer -= 1;
	}
	if(Etapa5 == 1)
	{
		Timer -= 1;
	}
	if(Etapa6 == 1)
	{
		Timer -= 1;
	}
	if(Etapa7 == 1)
	{
		Timer -= 1;
	}
	if(Etapa8 == 1)
	{
		Timer -= 1;
	}
	if(Etapa9 == 1)
	{
		Timer -= 1;
	}
}

static void Config_Timer1_Interrupt_1Hz(void)
{
	TCCR1A = 0x00;
	TCCR1C = 0x00;
	TCNT1 = 0x00;
	OCR1A = 15624;
	TIFR1 |= _BV(OCF1A);
	TIMSK1 |= _BV(OCIE1A);
	TCCR1B = (_BV(WGM12) | _BV(CS12) | _BV(CS10)) & (~_BV(WGM13) & ~_BV(CS11) & ~_BV(ICNC1) &
~_BV(ICES1));
}
```

Apêndice L – Código em C do código Ladder (método de equações lógicas) do 3º Exercício

```
#include "Funcoes_USART.h"
#include "Funcoes_ADC.h"
#include <avr/io.h>
#include <avr/interrupt.h>

static void Config_Timer1_Interrupt_1Hz(void);

uint8_t PFirstCycle = 1;
uint8_t IL = 0;
uint8_t Timer = 0;
uint8_t aux_Timer = 0;
uint8_t E0 = 0;
uint8_t E1 = 0;
uint8_t E2 = 0;
uint8_t E3 = 0;
uint8_t E4 = 0;
uint8_t E5 = 0;
uint8_t E6 = 0;
uint8_t E7 = 0;
uint8_t E8 = 0;
uint8_t E9 = 0;

int main (void)
{
	DDRD |= _BV(DDD7);
	PORTD &= ~_BV(PORTD7);
	DDRD |= _BV(DDD6);
	PORTD &= ~_BV(PORTD6);
	DDRD |= _BV(DDD5);
	PORTD &= ~_BV(PORTD5);
	DDRD |= _BV(DDD4);
	PORTD &= ~_BV(PORTD4);
	DDRD |= _BV(DDD3);
	PORTD &= ~_BV(PORTD3);
	DDRD |= _BV(DDD2);
	PORTD &= ~_BV(PORTD2);
	InicializarUSART();
	ImpedirInterruptUSART();
	InicializarADC();
	Config_Timer1_Interrupt_1Hz();
	sei();
	while(1)
	{
		if(E1 == 1)
		{
			aux_Timer = 0;
		}
		IL = (LerValor8bitADC(2)*0.4) + (IL*0.6);
		if((PFirstCycle == 1))
		{
			E0 = 1;
		}
		if((((E0 == 1) || ((E9 == 1) && (Timer == 0)) || ((E3 == 1) && (Timer == 0)) || ((E1 == 1))) && ((E4 == 0)
|| (E2 == 0))))
		{
```

```
            E1 = 1;
}
if(((E2 == 1) || ((E1 == 1) && (IL == 0))) && (E3 == 0))
{
            E2 = 1;
}
if(((E3 == 1) || ((E2 == 1) && (Timer == 0))) && (E1 == 0))
{
            E3 = 1;
}
if(((E4 == 1) || ((E1 == 1) && (IL == 255))) && (E5 == 0))
{
            E4 = 1;
}
if(((E5 == 1) || ((E4 == 1) && (Timer == 0))) && (E6 == 0))
{
            E5 = 1;
}
if(((E6 == 1) || ((E5 == 1) && (Timer == 0))) && (E7 == 0))
{
            E6 = 1;
}
if(((E7 == 1) || ((E6 == 1) && (Timer == 0))) && (E8 == 0))
{
            E7 = 1;
}
if(((E8 == 1) || ((E7 == 1) && (Timer == 0))) && (E9 == 0))
{
            E8 = 1;
}
if(((E9 == 1) || ((E8 == 1) && (Timer == 0))) && (E1 == 0))
{
            E9 = 1;
}
if((E2 == 1))
{
            if(aux_Timer == 0)
            {
                        Timer = 3;
                        aux_Timer = 1;
            }
            PORTD |= _BV(PORTD1);
            PORTD |= _BV(PORTD2);
            PORTD &= ~_BV(PORTD6);
            PORTD &= ~_BV(PORTD5);
            PORTD &= ~_BV(PORTD4);
            PORTD &= ~_BV(PORTD3);
}
if((E3 == 1))
{
            if(aux_Timer == 1)
            {
                        Timer = 3;
                        aux_Timer = 2;
            }
}
if((E4 == 1))
{
            if(aux_Timer == 2)
            {
                        Timer = 30;
                        aux_Timer = 3;
```

```
                }
                PORTD |= _BV(PORTD3);
                PORTD |= _BV(PORTD6);
                PORTD &= ~_BV(PORTD5);
                PORTD &= ~_BV(PORTD4);
                PORTD &= ~_BV(PORTD2);
                PORTD &= ~_BV(PORTD1);
        }
        if((E5 == 1))
        {
                if(aux_Timer == 3)
                {
                        Timer = 3;
                        aux_Timer = 4;
                }
                PORTD |= _BV(PORTD5);
                PORTD |= _BV(PORTD6);
                PORTD &= ~_BV(PORTD4);
                PORTD &= ~_BV(PORTD3);
                PORTD &= ~_BV(PORTD2);
                PORTD &= ~_BV(PORTD1);
        }
        if((E6 == 1))
        {
                if(aux_Timer == 4)
                {
                        Timer = 2;
                        aux_Timer = 5;
                }
                PORTD |= _BV(PORTD5);
                PORTD |= _BV(PORTD6);
                PORTD &= ~_BV(PORTD4);
                PORTD &= ~_BV(PORTD3);
                PORTD &= ~_BV(PORTD2);
                PORTD &= ~_BV(PORTD1);
        }
        if((E7 == 1))
        {
                if(aux_Timer == 5)
                {
                        Timer = 30;
                        aux_Timer = 6;
                }
                PORTD |= _BV(PORTD4);
                PORTD |= _BV(PORTD5);
                PORTD &= ~_BV(PORTD6);
                PORTD &= ~_BV(PORTD3);
                PORTD &= ~_BV(PORTD2);
                PORTD &= ~_BV(PORTD1);
        }
        if((E8 == 1))
        {
                if(aux_Timer == 6)
                {
                        Timer = 3;
                        aux_Timer = 7;
                }
                PORTD |= _BV(PORTD5);
                PORTD |= _BV(PORTD2);
                PORTD &= ~_BV(PORTD6);
                PORTD &= ~_BV(PORTD4);
                PORTD &= ~_BV(PORTD3);
```

```
                PORTD &= ~_BV(PORTD1);
        }
        if((E9 == 1))
        {
                if(aux_Timer == 7)
                {
                        Timer = 1;
                        aux_Timer = 8;
                }
                PORTD |= _BV(PORTD5);
                PORTD |= _BV(PORTD6);
                PORTD &= ~_BV(PORTD4);
                PORTD &= ~_BV(PORTD3);
                PORTD &= ~_BV(PORTD2);
                PORTD &= ~_BV(PORTD1);
        }
        if((E1 == 1))
        {
                E9 = 0;
                E3 = 0;
                E0 = 0;
        }
        if(((E2 == 1) || ((E4 == 1))))
        {
                E1 = 0;
        }
        if((E3 == 1))
        {
                E2 = 0;
        }
        if((E5 == 1))
        {
                E4 = 0;
        }
        if((E6 == 1))
        {
                E5 = 0;
        }
        if((E7 == 1))
        {
                E6 = 0;
        }
        if((E8 == 1))
        {
                E7 = 0;
        }
        if((E9 == 1))
        {
                E8 = 0;
        }
    }
}

ISR(TIMER1_COMPA_vect)
{
    if(E2 == 1)
    {
        Timer -= 1;
    }
    if(E3 == 1)
    {
        Timer -= 1;
```

```
		}
		if(E4 == 1)
		{
			Timer -= 1;
		}
		if(E5 == 1)
		{
			Timer -= 1;
		}
		if(E6 == 1)
		{
			Timer -= 1;
		}
		if(E7 == 1)
		{
			Timer -= 1;
		}
		if(E8 == 1)
		{
			Timer -= 1;
		}
		if(E9 == 1)
		{
			Timer -= 1;
		}
}

static void Config_Timer1_Interrupt_1Hz(void)
{
		TCCR1A = 0x00;
		TCCR1C = 0x00;
		TCNT1 = 0x00;
		OCR1A = 15624;
		TIFR1 |= _BV(OCF1A);
		TIMSK1 |= _BV(OCIE1A);
		TCCR1B = (_BV(WGM12) | _BV(CS12) | _BV(CS10)) & (~_BV(WGM13) & ~_BV(CS11) & ~_BV(ICNC1) &
~_BV(ICES1));
}
```

Apêndice M – Código em C do código Ladder (método de funções *Set/ Reset*) do 3º Exercício

```
#include "Funcoes_USART.h"
#include "Funcoes_ADC.h"
#include <avr/io.h>
#include <avr/interrupt.h>

static void Config_Timer1_Interrupt_1Hz(void);

uint8_t PFirstCycle = 1;
uint8_t IL = 0;
uint8_t Timer = 0;
uint8_t aux_Timer = 0;
uint8_t E0 = 0;
uint8_t E1 = 0;
uint8_t E2 = 0;
uint8_t E3 = 0;
uint8_t E4 = 0;
uint8_t E5 = 0;
uint8_t E6 = 0;
uint8_t E7 = 0;
uint8_t E8 = 0;
uint8_t E9 = 0;

int main (void)
{
	DDRD |= _BV(DDD7);
	PORTD &= ~_BV(PORTD7);
	DDRD |= _BV(DDD6);
	PORTD &= ~_BV(PORTD6);
	DDRD |= _BV(DDD5);
	PORTD &= ~_BV(PORTD5);
	DDRD |= _BV(DDD4);
	PORTD &= ~_BV(PORTD4);
	DDRD |= _BV(DDD3);
	PORTD &= ~_BV(PORTD3);
	DDRD |= _BV(DDD2);
	PORTD &= ~_BV(PORTD2);
	InicializarUSART();
	ImpedirInterruptUSART();
	InicializarADC();
	Config_Timer1_Interrupt_1Hz();
	sei();
	while(1)
	{
		if(E1 == 1)
		{
			aux_Timer = 0;
		}
		IL = (LerValor8bitADC(2)*0.4) + (IL*0.6);
		if((PFirstCycle == 1))
		{
			E0 = 1;
		}
		if((E0 == 1))
		{
			E0 = 0;
```

```
            E1 = 1;
}
if((E1 == 1) && (IL == 0))
{
            E1 = 0;
            E2 = 1;
}
if((E2 == 1) && (Timer == 0))
{
            E2 = 0;
            E3 = 1;
}
if((E3 == 1) && (Timer == 0))
{
            E3 = 0;
            E1 = 1;
}
if((E1 == 1) && (IL == 255))
{
            E1 = 0;
            E4 = 1;
}
if((E4 == 1) && (Timer == 0))
{
            E4 = 0;
            E5 = 1;
}
if((E5 == 1) && (Timer == 0))
{
            E5 = 0;
            E6 = 1;
}
if((E6 == 1) && (Timer == 0))
{
            E6 = 0;
            E7 = 1;
}
if((E7 == 1) && (Timer == 0))
{
            E7 = 0;
            E8 = 1;
}
if((E8 == 1) && (Timer == 0))
{
            E8 = 0;
            E9 = 1;
}
if((E9 == 1) && (Timer == 0))
{
            E9 = 0;
            E1 = 1;
}
if((E2 == 1))
{
            if(aux_Timer == 0)
            {
                        Timer = 3;
                        aux_Timer = 1;
            }
            PORTD |= _BV(PORTD1);
            PORTD |= _BV(PORTD2);
            PORTD &= ~_BV(PORTD6);
```

```
		PORTD &= ~_BV(PORTD5);
		PORTD &= ~_BV(PORTD4);
		PORTD &= ~_BV(PORTD3);
	}
	if((E3 == 1))
	{
		if(aux_Timer == 1)
		{
			Timer = 3;
			aux_Timer = 2;
		}
	}
	if((E4 == 1))
	{
		if(aux_Timer == 2)
		{
			Timer = 30;
			aux_Timer = 3;
		}
		PORTD |= _BV(PORTD3);
		PORTD |= _BV(PORTD6);
		PORTD &= ~_BV(PORTD5);
		PORTD &= ~_BV(PORTD4);
		PORTD &= ~_BV(PORTD2);
		PORTD &= ~_BV(PORTD1);
	}
	if((E5 == 1))
	{
		if(aux_Timer == 3)
		{
			Timer = 3;
			aux_Timer = 4;
		}
		PORTD |= _BV(PORTD1);
		PORTD |= _BV(PORTD6);
		PORTD &= ~_BV(PORTD5);
		PORTD &= ~_BV(PORTD4);
		PORTD &= ~_BV(PORTD3);
		PORTD &= ~_BV(PORTD2);
	}
	if((E6 == 1))
	{
		if(aux_Timer == 4)
		{
			Timer = 2;
			aux_Timer = 5;
		}
		PORTD |= _BV(PORTD5);
		PORTD |= _BV(PORTD6);
		PORTD &= ~_BV(PORTD4);
		PORTD &= ~_BV(PORTD3);
		PORTD &= ~_BV(PORTD2);
		PORTD &= ~_BV(PORTD1);
	}
	if((E7 == 1))
	{
		if(aux_Timer == 5)
		{
			Timer = 30;
			aux_Timer = 6;
		}
		PORTD |= _BV(PORTD5);
```

```
                PORTD |= _BV(PORTD4);
                PORTD &= ~_BV(PORTD6);
                PORTD &= ~_BV(PORTD3);
                PORTD &= ~_BV(PORTD2);
                PORTD &= ~_BV(PORTD1);
        }
        if((E8 == 1))
        {
                if(aux_Timer == 6)
                {
                        Timer = 3;
                        aux_Timer = 7;
                }
                PORTD |= _BV(PORTD5);
                PORTD |= _BV(PORTD2);
                PORTD &= ~_BV(PORTD6);
                PORTD &= ~_BV(PORTD4);
                PORTD &= ~_BV(PORTD3);
                PORTD &= ~_BV(PORTD1);
        }
        if((E9 == 1))
        {
                if(aux_Timer == 7)
                {
                        Timer = 2;
                        aux_Timer = 8;
                }
                PORTD |= _BV(PORTD5);
                PORTD |= _BV(PORTD6);
                PORTD &= ~_BV(PORTD4);
                PORTD &= ~_BV(PORTD3);
                PORTD &= ~_BV(PORTD2);
                PORTD &= ~_BV(PORTD1);
        }
    }
}

ISR(TIMER1_COMPA_vect)
{
    if(E2 == 1)
    {
        Timer -= 1;
    }
    if(E3 == 1)
    {
        Timer -= 1;
    }
    if(E4 == 1)
    {
        Timer -= 1;
    }
    if(E5 == 1)
    {
        Timer -= 1;
    }
    if(E6 == 1)
    {
        Timer -= 1;
    }
    if(E7 == 1)
    {
        Timer -= 1;
```

```
		}
		if(E8 == 1)
		{
			Timer -= 1;
		}
		if(E9 == 1)
		{
			Timer -= 1;
		}
}

static void Config_Timer1_Interrupt_1Hz(void)
{
		TCCR1A = 0x00;
		TCCR1C = 0x00;
		TCNT1 = 0x00;
		OCR1A = 15624;
		TIFR1 |= _BV(OCF1A);
		TIMSK1 |= _BV(OCIE1A);
		TCCR1B = (_BV(WGM12) | _BV(CS12) | _BV(CS10)) & (~_BV(WGM13) & ~_BV(CS11) & ~_BV(ICNC1) &
~_BV(ICES1));
}
```

Apêndice N – Código em C do código em Grafcet do 4º Exercício

```
#include "Funcoes_USART.h"
#include "Funcoes_ADC.h"
#include <avr/io.h>
#include <avr/interrupt.h>

int main (void)
{
	DDRD |= _BV(DDD5);
	PORTD &= ~_BV(PORTD5);
	DDRD |= _BV(DDD4);
	PORTD &= ~_BV(PORTD4);
	DDRD |= _BV(DDD3);
	PORTD &= ~_BV(PORTD3);
	DDRD |= _BV(DDD2);
	PORTD &= ~_BV(PORTD2);
	DDRD |= _BV(DDD1);
	PORTD &= ~_BV(PORTD1);
	DDRD |= _BV(DDD0);
	PORTD &= ~_BV(PORTD0);
	uint8_t Etapa0 = 1;
	uint8_t Etapa1 = 0;
	uint8_t Etapa2 = 0;
	uint8_t Etapa3 = 0;
	uint8_t Etapa4 = 0;
	uint8_t Etapa5 = 0;
	uint8_t Etapa6 = 0;
	uint8_t Etapa7 = 0;
	uint8_t S0 = 0;
	uint8_t S1 = 0;
	uint8_t S2 = 0;
	uint8_t S3 = 0;
	uint8_t A1 = 0;
	uint8_t A0 = 0;
	InicializarUSART();
	ImpedirInterruptUSART();
	InicializarADC();
	sei();
	while(1)
	{
		S0 = (LerValor8bitADC(2)*0.4) + (S0*0.6);
		S1 = (LerValor8bitADC(3)*0.4) + (S1*0.6);
		S2 = (LerValor8bitADC(4)*0.4) + (S2*0.6);
		S3 = (LerValor8bitADC(5)*0.4) + (S3*0.6);
		A1 = (LerValor8bitADC(6)*0.4) + (A1*0.6);
		A0 = (LerValor8bitADC(7)*0.4) + (A0*0.6);
		if(Etapa0 == 1)
		{
			Etapa1 = 1;
			Etapa0 = 0;
		}
		if(Etapa1 == 1)
		{
			PORTD &= ~_BV(PORTD5);
			PORTD &= ~_BV(PORTD4);
			PORTD &= ~_BV(PORTD3);
			PORTD &= ~_BV(PORTD2);
			PORTD &= ~_BV(PORTD1);
```

```
            PORTD &= ~_BV(PORTD0);
}
if(((S0<50)&&(S1<50)&&(S2>220)) && Etapa1 == 1)
{
            Etapa2 = 1;
            Etapa1 = 0;
}
if(Etapa2 == 1)
{
            PORTD &= ~_BV(PORTD5);
            PORTD &= ~_BV(PORTD4);
            PORTD |= _BV(PORTD3);
            PORTD |= _BV(PORTD2);
            PORTD &= ~_BV(PORTD1);
            PORTD &= ~_BV(PORTD0);
}
if(((S1>220)) && Etapa2 == 1)
{
            Etapa3 = 1;
            Etapa2 = 0;
}
if(Etapa3 == 1)
{
            PORTD &= ~_BV(PORTD5);
            PORTD &= ~_BV(PORTD4);
            PORTD &= ~_BV(PORTD3);
            PORTD &= ~_BV(PORTD2);
            PORTD &= ~_BV(PORTD1);
            PORTD &= ~_BV(PORTD0);
}
if(((S1<50)) && Etapa3 == 1)
{
            Etapa2 = 1;
            Etapa3 = 0;
}
if(((S1<50)&&(S3>220)&&(A0<50)) && Etapa2 == 1)
{
            Etapa4 = 1;
            Etapa2 = 0;
}
if(Etapa4 == 1)
{
            PORTD |= _BV(PORTD5);
            PORTD &= ~_BV(PORTD4);
            PORTD &= ~_BV(PORTD3);
            PORTD &= ~_BV(PORTD2);
            PORTD &= ~_BV(PORTD1);
            PORTD &= ~_BV(PORTD0);
}
if(((S1>220)) && Etapa4 == 1)
{
            Etapa5 = 1;
            Etapa4 = 0;
}
if(Etapa5 == 1)
{
            PORTD &= ~_BV(PORTD5);
            PORTD &= ~_BV(PORTD4);
            PORTD &= ~_BV(PORTD3);
            PORTD &= ~_BV(PORTD2);
            PORTD &= ~_BV(PORTD1);
            PORTD &= ~_BV(PORTD0);
```

```
                }
                if(((S1<50)) && Etapa5 == 1)
                {
                        Etapa4 = 1;
                        Etapa5 = 0;
                }
                if(((A1>220)&&(S1<50)) && Etapa4 == 1)
                {
                        Etapa6 = 1;
                        Etapa4 = 0;
                }
                if(Etapa6 == 1)
                {
                        PORTD &= ~_BV(PORTD5);
                        PORTD |= _BV(PORTD4);
                        PORTD &= ~_BV(PORTD3);
                        PORTD &= ~_BV(PORTD2);
                        PORTD &= ~_BV(PORTD1);
                        PORTD &= ~_BV(PORTD0);
                }
                if(((S1>220)) && Etapa6 == 1)
                {
                        Etapa7 = 1;
                        Etapa6 = 0;
                }
                if(Etapa7 == 1)
                {
                        PORTD &= ~_BV(PORTD5);
                        PORTD &= ~_BV(PORTD4);
                        PORTD &= ~_BV(PORTD3);
                        PORTD &= ~_BV(PORTD2);
                        PORTD &= ~_BV(PORTD1);
                        PORTD &= ~_BV(PORTD0);
                }
                if(((S1<50)) && Etapa7 == 1)
                {
                        Etapa6 = 1;
                        Etapa7 = 0;
                }
                if(((A0>220)&&(S1<50)) && ((S0<50)) && Etapa6 == 1)
                {
                        Etapa1 = 1;
                        Etapa6 = 0;
                }
        }
}
```

Apêndice O – Código em C do código Ladder (método de equações lógicas) do 4º Exercício

```
#include "Funcoes_USART.h"
#include "Funcoes_ADC.h"
#include <avr/io.h>
#include <avr/interrupt.h>

uint8_t PFirstCycle = 1;
uint8_t S0 = 0;
uint8_t S1 = 0;
uint8_t S2 = 0;
uint8_t S3 = 0;
uint8_t A1 = 0;
uint8_t A0 = 0;
uint8_t E0 = 0;
uint8_t E1 = 0;
uint8_t E2 = 0;
uint8_t E3 = 0;
uint8_t E4 = 0;
uint8_t E5 = 0;
uint8_t E6 = 0;
uint8_t E7 = 0;

int main (void)
{
	DDRD |= _BV(DDD7);
	PORTD &= ~_BV(PORTD7);
	DDRD |= _BV(DDD6);
	PORTD &= ~_BV(PORTD6);
	DDRD |= _BV(DDD5);
	PORTD &= ~_BV(PORTD5);
	DDRD |= _BV(DDD4);
	PORTD &= ~_BV(PORTD4);
	DDRD |= _BV(DDD3);
	PORTD &= ~_BV(PORTD3);
	DDRD |= _BV(DDD2);
	PORTD &= ~_BV(PORTD2);
	InicializarUSART();
	ImpedirInterruptUSART();
	InicializarADC();
	sei();
	while(1)
	{
		S0 = (LerValor8bitADC(2)*0.4) + (S0*0.6);
		S1 = (LerValor8bitADC(3)*0.4) + (S1*0.6);
		S2 = (LerValor8bitADC(4)*0.4) + (S2*0.6);
		S3 = (LerValor8bitADC(5)*0.4) + (S3*0.6);
		A1 = (LerValor8bitADC(6)*0.4) + (A1*0.6);
		A0 = (LerValor8bitADC(7)*0.4) + (A0*0.6);
		if((PFirstCycle == 1))
		{
			E0 = 1;
		}
		if(((E0 == 1) || ((E6 == 1) && (A0 == 255) && (S0 == 0) && (S1 == 0)) || ((E1 == 1))) && (E2 == 0))
		{
			E1 = 1;
		}
```

```
		if(((E2 == 1) || ((E3 == 1) && (S1 == 0)) || ((E1 == 1) && (S0 == 0) && (S1 == 0) && (S2 == 255))) &&
(E4 == 0))
		{
			E2 = 1;
		}
		if(((E3 == 1) || ((E2 == 1) && (S1 == 255))) && (E4 == 0))
		{
			E3 = 1;
		}
		if(((E4 == 1) || ((E5 == 1) && (S1 == 0)) || ((E2 == 1) && (S1 == 0) && (S3 == 255) && (A0 == 0))) &&
(E6 == 0))
		{
			E4 = 1;
		}
		if(((E5 == 1) || ((E4 == 1) && (S1 == 255))) && (E6 == 0))
		{
			E5 = 1;
		}
		if(((E6 == 1) || ((E7 == 1) && (S1 == 0)) || ((E4 == 1) && (A1 == 255) && (S1 == 0))) && (E1 == 0))
		{
			E6 = 1;
		}
		if(((E7 == 1) || ((E6 == 1) && (S1 == 255))) && (E1 == 0))
		{
			E7 = 1;
		}
		if((E2 == 1))
		{
			PORTD |= _BV(PORTD3);
			PORTD |= _BV(PORTD4);
			PORTD &= ~_BV(PORTD6);
			PORTD &= ~_BV(PORTD5);
			PORTD &= ~_BV(PORTD2);
			PORTD &= ~_BV(PORTD1);
		}
		if((E4 == 1))
		{
			PORTD |= _BV(PORTD6);
			PORTD &= ~_BV(PORTD5);
			PORTD &= ~_BV(PORTD4);
			PORTD &= ~_BV(PORTD3);
			PORTD &= ~_BV(PORTD2);
			PORTD &= ~_BV(PORTD1);
		}
		if((E6 == 1))
		{
			PORTD |= _BV(PORTD5);
			PORTD &= ~_BV(PORTD6);
			PORTD &= ~_BV(PORTD4);
			PORTD &= ~_BV(PORTD3);
			PORTD &= ~_BV(PORTD2);
			PORTD &= ~_BV(PORTD1);
		}
		if((E1 == 1))
		{
			E0 = 0;
		}
		if((E2 == 1))
		{
			E1 = 0;
		}
		if(((E3 == 1) || ((E4 == 1))))
```

```
                {
                        E2 = 0;
                }
                if((E2 == 1))
                {
                        E3 = 0;
                }
                if(((E5 == 1) || ((E6 == 1))))
                {
                        E4 = 0;
                }
                if((E4 == 1))
                {
                        E5 = 0;
                }
                if(((E1 == 1) || ((E7 == 1))))
                {
                        E6 = 0;
                }
                if((E6 == 1))
                {
                        E7 = 0;
                }
        }
}
```

Apêndice P – Código em C do código Ladder (método de funções *Set/ Reset*) do 4º Exercício

```
#include "Funcoes_USART.h"
#include "Funcoes_ADC.h"
#include <avr/io.h>
#include <avr/interrupt.h>
uint8_t PFirstCycle = 1;
uint8_t S0 = 0;
uint8_t S1 = 0;
uint8_t S2 = 0;
uint8_t S3 = 0;
uint8_t A1 = 0;
uint8_t A0 = 0;
uint8_t E0 = 0;
uint8_t E1 = 0;
uint8_t E2 = 0;
uint8_t E3 = 0;
uint8_t E4 = 0;
uint8_t E5 = 0;
uint8_t E6 = 0;
uint8_t E7 = 0;

int main (void)
{
	DDRD |= _BV(DDD7);
	PORTD &= ~_BV(PORTD7);
	DDRD |= _BV(DDD6);
	PORTD &= ~_BV(PORTD6);
	DDRD |= _BV(DDD5);
	PORTD &= ~_BV(PORTD5);
	DDRD |= _BV(DDD4);
	PORTD &= ~_BV(PORTD4);
	DDRD |= _BV(DDD3);
	PORTD &= ~_BV(PORTD3);
	DDRD |= _BV(DDD2);
	PORTD &= ~_BV(PORTD2);
	InicializarUSART();
	ImpedirInterruptUSART();
	InicializarADC();
	sei();
	while(1)
	{
		S0 = (LerValor8bitADC(2)*0.4) + (S0*0.6);
		S1 = (LerValor8bitADC(3)*0.4) + (S1*0.6);
		S2 = (LerValor8bitADC(4)*0.4) + (S2*0.6);
		S3 = (LerValor8bitADC(5)*0.4) + (S3*0.6);
		A1 = (LerValor8bitADC(6)*0.4) + (A1*0.6);
		A0 = (LerValor8bitADC(7)*0.4) + (A0*0.6);
		if((PFirstCycle == 1))
		{
			E0 = 1;
		}
		if((E0 == 1))
		{
			E0 = 0;
			E1 = 1;
		}
```

```
if((E1 == 1) && (S0 == 0) && (S1 == 0) && (S2 == 255))
{
        E1 = 0;
        E2 = 1;
}
if((E2 == 1) && (S1 == 255))
{
        E2 = 0;
        E3 = 1;
}
if((E3 == 1) && (S1 == 0))
{
        E3 = 0;
        E2 = 1;
}
if((E2 == 1) && (S1 == 0) && (S3 == 255) && (A0 == 0))
{
        E2 = 0;
        E4 = 1;
}
if((E4 == 1) && (S1 == 255))
{
        E4 = 0;
        E5 = 1;
}
if((E5 == 1) && (S1 == 0))
{
        E5 = 0;
        E4 = 1;
}
if((E4 == 1) && (A1 == 255) && (S1 == 0))
{
        E4 = 0;
        E6 = 1;
}
if((E6 == 1) && (S1 == 255))
{
        E6 = 0;
        E7 = 1;
}
if((E7 == 1) && (S1 == 0))
{
        E7 = 0;
        E6 = 1;
}
if((E6 == 1) && (A0 == 255) && (S1 == 0) && (S0 == 0))
{
        E6 = 0;
        E1 = 1;
}
if((E2 == 1))
{
        PORTD |= _BV(PORTD3);
        PORTD |= _BV(PORTD4);
        PORTD &= ~_BV(PORTD6);
        PORTD &= ~_BV(PORTD5);
        PORTD &= ~_BV(PORTD2);
        PORTD &= ~_BV(PORTD1);
}
if((E4 == 1))
{
        PORTD |= _BV(PORTD6);
```

```
                        PORTD &= ~_BV(PORTD5);
                        PORTD &= ~_BV(PORTD4);
                        PORTD &= ~_BV(PORTD3);
                        PORTD &= ~_BV(PORTD2);
                        PORTD &= ~_BV(PORTD1);
                }
                if((E6 == 1))
                {
                        PORTD |= _BV(PORTD5);
                        PORTD &= ~_BV(PORTD6);
                        PORTD &= ~_BV(PORTD4);
                        PORTD &= ~_BV(PORTD3);
                        PORTD &= ~_BV(PORTD2);
                        PORTD &= ~_BV(PORTD1);
                }
        }
}
```

Apêndice Q – Código em C do código em Grafcet do 5º Exercício

```
#include "Funcoes_USART.h"
#include "Funcoes_ADC.h"
#include <avr/io.h>
#include <avr/interrupt.h>
int main (void)
{
	DDRD |= _BV(DDD5);
	PORTD &= ~_BV(PORTD5);
	DDRD |= _BV(DDD4);
	PORTD &= ~_BV(PORTD4);
	DDRD |= _BV(DDD3);
	PORTD &= ~_BV(PORTD3);
	DDRD |= _BV(DDD2);
	PORTD &= ~_BV(PORTD2);
	uint8_t Etapa0 = 1;
	uint8_t Etapa1 = 0;
	uint8_t Etapa2 = 0;
	uint8_t Etapa3 = 0;
	uint8_t Etapa4 = 0;
	uint8_t Etapa5 = 0;
	uint8_t Start = 0;
	uint8_t SE1 = 0;
	uint8_t SE2 = 0;
	uint8_t Stop = 0;
	uint8_t Cnt = 0;
	InicializarUSART();
	ImpedirInterruptUSART();
	InicializarADC();
	sei();
	while(1)
	{
		Start = (LerValor8bitADC(2)*0.4) + (Start*0.6);
		SE1 = (LerValor8bitADC(3)*0.4) + (SE1*0.6);
		SE2 = (LerValor8bitADC(4)*0.4) + (SE2*0.6);
		Stop = (LerValor8bitADC(5)*0.4) + (Stop*0.6);
		if(Etapa0 == 1)
		{
			Etapa1 = 1;
			Etapa0 = 0;
		}
		if(Etapa1 == 1)
		{
			PORTD &= ~_BV(PORTD5);
			PORTD &= ~_BV(PORTD4);
			PORTD &= ~_BV(PORTD3);
			PORTD &= ~_BV(PORTD2);
		}
		if(((Start>220)&&(Stop<50)) && Etapa1 == 1)
		{
			Etapa2 = 1;
			Etapa1 = 0;
		}
		if(Etapa2 == 1)
		{
			PORTD |= _BV(PORTD5);
			PORTD &= ~_BV(PORTD4);
			PORTD &= ~_BV(PORTD3);
```

```
                PORTD &= ~_BV(PORTD2);
        }
        if(((Stop>220)) && Etapa2 == 1)
        {
                Etapa1 = 1;
                Etapa2 = 0;
        }
        if(((SE2>220)&&(Stop<50)) && Etapa2 == 1)
        {
                Etapa3 = 1;
                Etapa2 = 0;
        }
        if(Etapa3 == 1)
        {
                PORTD &= ~_BV(PORTD5);
                PORTD |= _BV(PORTD4);
                PORTD &= ~_BV(PORTD3);
                PORTD &= ~_BV(PORTD2);
                Cnt=0;
        }
        if(((SE1>220)) && Etapa3 == 1)
        {
                Etapa4 = 1;
                Etapa3 = 0;
        }
        if(Etapa4 == 1)
        {
                PORTD &= ~_BV(PORTD5);
                PORTD |= _BV(PORTD4);
                PORTD &= ~_BV(PORTD3);
                PORTD &= ~_BV(PORTD2);
                Cnt+=1;
        }
        if(Etapa4 == 1)
        {
                Etapa5 = 1;
                Etapa4 = 0;
        }
        if(Etapa5 == 1)
        {
                PORTD &= ~_BV(PORTD5);
                PORTD |= _BV(PORTD4);
                PORTD &= ~_BV(PORTD3);
                PORTD &= ~_BV(PORTD2);
        }
        if(((Cnt!=10)&&(SE1>220)&&(Stop<50)) && Etapa5 == 1)
        {
                Etapa4 = 1;
                Etapa5 = 0;
        }
        if(((Stop>220)) && Etapa5 == 1)
        {
                Etapa1 = 1;
                Etapa5 = 0;
        }
        if(((Cnt==10)&&(Stop<50)) && Etapa5 == 1)
        {
                Etapa2 = 1;
                Etapa5 = 0;
        }
    }
}
```

Apêndice R – Código em C do código Ladder (método de equações lógicas) do 5º Exercício

```
#include "Funcoes_USART.h"
#include "Funcoes_ADC.h"
#include <avr/io.h>
#include <avr/interrupt.h>

uint8_t PFirstCycle = 1;
uint8_t Start = 0;
uint8_t SE1 = 0;
uint8_t SE2 = 0;
uint8_t Stop = 0;
uint8_t CNT = 0;
uint8_t E0 = 0;
uint8_t E1 = 0;
uint8_t E2 = 0;
uint8_t E3 = 0;
uint8_t E4 = 0;
uint8_t E5 = 0;

int main (void)
{
	DDRD |= _BV(DDD7);
	PORTD &= ~_BV(PORTD7);
	DDRD |= _BV(DDD6);
	PORTD &= ~_BV(PORTD6);
	DDRD |= _BV(DDD5);
	PORTD &= ~_BV(PORTD5);
	DDRD |= _BV(DDD4);
	PORTD &= ~_BV(PORTD4);
	InicializarUSART();
	ImpedirInterruptUSART();
	InicializarADC();
	sei();
	while(1)
	{
		Start = (LerValor8bitADC(2)*0.4) + (Start*0.6);
		SE1 = (LerValor8bitADC(3)*0.4) + (SE1*0.6);
		SE2 = (LerValor8bitADC(4)*0.4) + (SE2*0.6);
		Stop = (LerValor8bitADC(5)*0.4) + (Stop*0.6);
		if((PFirstCycle == 1))
		{
			E0 = 1;
		}
		if(((E1 == 1) || ((E5 == 1) && (Stop == 255)) || ((E2 == 1) && (Stop == 255))) && (E2 == 0))
		{
			E1 = 1;
		}
		if(((E2 == 1) || ((E5 == 1) && (CNT == 10) && (Stop == 0)) || ((E1 == 1) && (Start == 255) && (Stop ==
0))) && (E3 == 0))
		{
			E2 = 1;
		}
		if(((E3 == 1) || ((E2 == 1) && (SE2 == 255) && (Stop == 0))) && (E4 == 0))
		{
			E3 = 1;
		}
```

```
                    if(((E4 == 1) || ((E5 == 1) && (CNT != 10) && (SE1 == 255) && (Stop == 0)) || ((E3 == 1) && (SE1 ==
255))) && (E1 == 0))
                    {
                              E4 = 1;
                    }
                    if(((E5 == 1) || ((E4 == 1))) && (E2 == 0) && (E1 == 0))
                    {
                              E5 = 1;
                    }
                    if((E2 == 1))
                    {
                              PORTD |= _BV(PORTD6);
                              PORTD &= ~_BV(PORTD5);
                              PORTD &= ~_BV(PORTD4);
                              PORTD &= ~_BV(PORTD3);
                    }
                    if(((E3 == 1) || ((E5 == 1)) || ((E4 == 1))))
                    {
                              PORTD |= _BV(PORTD5);
                              PORTD &= ~_BV(PORTD6);
                              PORTD &= ~_BV(PORTD4);
                              PORTD &= ~_BV(PORTD3);
                    }
                    if((E3 == 1))
                    {
                              CNT = 0;
                    }
                    if((E4 == 1) && (SE1 == 255))
                    {
                              CNT += 1;
                    }
                    if((E1 == 1))
                    {
                              E0 = 0;
                    }
                    if((E2 == 1))
                    {
                              E1 = 0;
                    }
                    if(((E1 == 1) || ((E3 == 1))))
                    {
                              E2 = 0;
                    }
                    if((E4 == 1))
                    {
                              E3 = 0;
                    }
                    if((E5 == 1))
                    {
                              E4 = 0;
                    }
                    if(((E1 == 1) || ((E2 == 1))))
                    {
                              E5 = 0;
                    }
          }
}
```

Apêndice S – Código em C do código Ladder (método de funções *Set/ Reset*) do 5º Exercício

```
#include "Funcoes_USART.h"
#include "Funcoes_ADC.h"
#include <avr/io.h>
#include <avr/interrupt.h>

uint8_t PFirstCycle = 1;
uint8_t Start = 0;
uint8_t SE1 = 0;
uint8_t SE2 = 0;
uint8_t Stop = 0;
uint8_t CNT = 0;
uint8_t E0 = 0;
uint8_t E1 = 0;
uint8_t E2 = 0;
uint8_t E3 = 0;
uint8_t E4 = 0;
uint8_t E5 = 0;

int main (void)
{
	DDRD |= _BV(DDD7);
	PORTD &= ~_BV(PORTD7);
	DDRD |= _BV(DDD6);
	PORTD &= ~_BV(PORTD6);
	DDRD |= _BV(DDD5);
	PORTD &= ~_BV(PORTD5);
	DDRD |= _BV(DDD4);
	PORTD &= ~_BV(PORTD4);
	InicializarUSART();
	ImpedirInterruptUSART();
	InicializarADC();
	sei();
	while(1)
	{
		Start = (LerValor8bitADC(2)*0.4) + (Start*0.6);
		SE1 = (LerValor8bitADC(3)*0.4) + (SE1*0.6);
		SE2 = (LerValor8bitADC(4)*0.4) + (SE2*0.6);
		Stop = (LerValor8bitADC(5)*0.4) + (Stop*0.6);
		if((PFirstCycle == 1))
		{
			E0 = 1;
		}
		if((E0 == 1))
		{
			E0 = 0;
			E1 = 1;
		}
		if((E1 == 1) && (Start == 255) && (Stop == 0))
		{
			E1 = 0;
			E2 = 1;
		}
		if((E2 == 1) && (Stop == 255))
		{
			E2 = 0;
```

```
                E1 = 1;
        }
        if((E2 == 1) && (SE2 == 255) && (Stop == 0))
        {
                E2 = 0;
                E3 = 1;
        }
        if((E3 == 1) && (SE1 == 255))
        {
                E3 = 0;
                E4 = 1;
        }
        if((E4 == 1) && (SE1 == 0))
        {
                E4 = 0;
                E5 = 1;
        }
        if((E5 == 1) && (Stop == 255))
        {
                E5 = 0;
                E1 = 1;
        }
        if((E5 == 1) && (CNT != 10) && (SE1 == 255) && (Stop == 0))
        {
                E5 = 0;
                E4 = 1;
        }
        if((E5 == 1) && (CNT == 10) && (Stop == 0))
        {
                E5 = 0;
                E2 = 1;
        }
        if((E2 == 1))
        {
                PORTD |= _BV(PORTD6);
                PORTD &= ~_BV(PORTD5);
                PORTD &= ~_BV(PORTD4);
                PORTD &= ~_BV(PORTD3);
        }
        if(((E3 == 1) || ((E5 == 1)) || ((E4 == 1))))
        {
                PORTD |= _BV(PORTD5);
                PORTD &= ~_BV(PORTD6);
                PORTD &= ~_BV(PORTD4);
                PORTD &= ~_BV(PORTD3);
        }
        if((E3 == 1))
        {
                CNT = 0;
        }
        if((E4 == 1) && (SE1 == 255))
        {
                CNT += 1;
        }
    }
}
```

Apêndice T – Código em C do código em Grafcet do 6º Exercício

```
#include "Funcoes_USART.h"
#include "Funcoes_ADC.h"
#include <avr/io.h>
#include <avr/interrupt.h>

static void Config_Timer1_Interrupt_1Hz(void);

int main (void)
{
	DDRD |= _BV(DDD5);
	PORTD &= ~_BV(PORTD5);
	DDRD |= _BV(DDD4);
	PORTD &= ~_BV(PORTD4);
	DDRD |= _BV(DDD3);
	PORTD &= ~_BV(PORTD3);
	DDRD |= _BV(DDD2);
	PORTD &= ~_BV(PORTD2);
	DDRD |= _BV(DDD1);
	PORTD &= ~_BV(PORTD1);
	DDRD |= _BV(DDD0);
	PORTD &= ~_BV(PORTD0);
	uint8_t Etapa0 = 1;
	uint8_t Etapa1 = 0;
	uint8_t Etapa2 = 0;
	uint8_t Etapa3 = 0;
	uint8_t Etapa4 = 0;
	uint8_t Etapa5 = 0;
	uint8_t Etapa6 = 0;
	uint8_t Etapa7 = 0;
	uint8_t Etapa8 = 0;
	uint8_t Etapa9 = 0;
	uint8_t Etapa10 = 0;
	uint8_t Etapa11 = 0;
	uint8_t Etapa12 = 0;
	uint8_t Dcy = 0;
	uint8_t Dp = 0;
	uint8_t B = 0;
	uint8_t S0 = 0;
	uint8_t S1 = 0;
	InicializarUSART();
	ImpedirInterruptUSART();
	InicializarADC();
	Config_Timer1_Interrupt_1Hz();
	sei();
	while(1)
	{
		Dcy = (LerValor8bitADC(2)*0.4) + (Dcy*0.6);
		Dp = (LerValor8bitADC(3)*0.4) + (Dp*0.6);
		B = (LerValor8bitADC(4)*0.4) + (B*0.6);
		S0 = (LerValor8bitADC(5)*0.4) + (S0*0.6);
		S1 = (LerValor8bitADC(6)*0.4) + (S1*0.6);
		if(Etapa0 == 1)
		{
			Etapa1 = 1;
			Etapa0 = 0;
		}
		if(Etapa1 == 1)
		{
```

```
                PORTD &= ~_BV(PORTD5);
                PORTD &= ~_BV(PORTD4);
                PORTD &= ~_BV(PORTD3);
                PORTD &= ~_BV(PORTD2);
                PORTD &= ~_BV(PORTD1);
                PORTD &= ~_BV(PORTD0);
                Timer=15;
        }
        if(((B<50)) && ((Dcy>220)) && Etapa1 == 1)
        {
                Etapa2 = 1;
        }
        if(Etapa2 == 1)
        {
                PORTD |= _BV(PORTD5);
                PORTD &= ~_BV(PORTD4);
                PORTD &= ~_BV(PORTD3);
                PORTD &= ~_BV(PORTD2);
                PORTD &= ~_BV(PORTD1);
                PORTD &= ~_BV(PORTD0);
        }
        if(((B==150)) && Etapa2 == 1)
        {
                Etapa3 = 1;
                Etapa2 = 0;
                PORTD &= ~_BV(PORTD5);
        }
        if(Etapa3 == 1)
        {
                PORTD |= _BV(PORTD5);
        }
        if(((B<50)) && Etapa3 == 1)
        {
                Etapa4 = 1;
                Etapa3 = 0;
                PORTD &= ~_BV(PORTD5);
        }
        if(Etapa4 == 1)
        {
                PORTD |= _BV(PORTD5);
        }
        if(((B==200)) && Etapa4 == 1)
        {
                Etapa5 = 1;
                Etapa4 = 0;
                PORTD &= ~_BV(PORTD5);
        }
        if(Etapa5 == 1)
        {
                PORTD |= _BV(PORTD5);
        }
        if(((B<50)) && Etapa5 == 1)
        {
                Etapa6 = 1;
                Etapa5 = 0;
                PORTD &= ~_BV(PORTD5);
        }
        if(Etapa6 == 1)
        {
        }
        if(((Dp<50)) && ((Dcy>220)) && Etapa1 == 1)
        {
```

```
            Etapa7 = 1;
            Etapa1 = 0;
}
if(Etapa7 == 1)
{
            PORTD |= _BV(PORTD5);
}
if(((Dp>220)) && Etapa7 == 1)
{
            Etapa8 = 1;
            Etapa7 = 0;
            PORTD &= ~_BV(PORTD5);
}
if(Etapa8 == 1)
{
            PORTD |= _BV(PORTD5);
}
if(((Dp>220)) && Etapa8 == 1)
{
            Etapa9 = 1;
            Etapa8 = 0;
            PORTD &= ~_BV(PORTD5);
}
if(Etapa9 == 1)
{
}
if((1) && Etapa6 == 1 && Etapa9 == 1)
{
            Etapa10 = 1;
            Etapa6 = 0;
            Etapa9 = 0;
}
if(Etapa10 == 1)
{
            PORTD &= ~_BV(PORTD5);
            PORTD &= ~_BV(PORTD4);
            PORTD &= ~_BV(PORTD3);
            PORTD &= ~_BV(PORTD2);
            PORTD |= _BV(PORTD1);
            PORTD &= ~_BV(PORTD0);
}
if(((Timer==0)&&(S0>220)) && Etapa10 == 1)
{
            Etapa11 = 1;
            Etapa10 = 0;
}
if(Etapa11 == 1)
{
            PORTD &= ~_BV(PORTD5);
            PORTD &= ~_BV(PORTD4);
            PORTD &= ~_BV(PORTD3);
            PORTD &= ~_BV(PORTD2);
            PORTD &= ~_BV(PORTD1);
            PORTD |= _BV(PORTD0);
}
if(((S1>220)) && Etapa11 == 1)
{
            Etapa12 = 1;
            Etapa11 = 0;
}
if(Etapa12 == 1)
{
```

```
                                PORTD &= ~_BV(PORTD5);
                                PORTD &= ~_BV(PORTD4);
                                PORTD &= ~_BV(PORTD3);
                                PORTD &= ~_BV(PORTD2);
                                PORTD &= ~_BV(PORTD1);
                                PORTD &= ~_BV(PORTD0);
                        }
                        if(((S0>220)) && Etapa12 == 1)
                        {
                                Etapa1 = 1;
                                Etapa12 = 0;
                        }
          }
}

ISR(TIMER1_COMPA_vect)
{
        if(Etapa10 == 1)
        {
           Timer -= 1;
        }
}

static void Config_Timer1_Interrupt_1Hz(void)
{
        TCCR1A = 0x00;
        TCCR1C = 0x00;
        TCNT1 = 0x00;
        OCR1A = 15624;
        TIFR1 |= _BV(OCF1A);
        TIMSK1 |= _BV(OCIE1A);
        TCCR1B = (_BV(WGM12) | _BV(CS12) | _BV(CS10)) & (~_BV(WGM13) & ~_BV(CS11) & ~_BV(ICNC1) &
~_BV(ICES1));
}
```

Apêndice U – Código em C do código Ladder (método de equações lógicas) do 6º Exercício

```
#include "Funcoes_USART.h"
#include "Funcoes_ADC.h"
#include <avr/io.h>
#include <avr/interrupt.h>

static void Config_Timer1_Interrupt_1Hz(void);

uint8_t PFirstCycle = 1;
uint8_t Dcy = 0;
uint8_t Dp = 0;
uint8_t B = 0;
uint8_t S0 = 0;
uint8_t S1 = 0;
uint8_t Temp = 0;
uint8_t aux_Temp = 0;
uint8_t a = 0;
uint8_t b = 0;
uint8_t E0 = 0;
uint8_t E1 = 0;
uint8_t E2 = 0;
uint8_t E3 = 0;
uint8_t E4 = 0;
uint8_t E5 = 0;
uint8_t E6 = 0;
uint8_t E7 = 0;
uint8_t E8 = 0;
uint8_t E9 = 0;
uint8_t E10 = 0;
uint8_t E11 = 0;
uint8_t E12 = 0;
uint8_t BMenorQuea = 0;
uint8_t BIgualAa = 0;
uint8_t BMaiorQuea = 0;
uint8_t BMenorQueb = 0;
uint8_t BIgualAb = 0;
uint8_t BMaiorQueb = 0;

int main (void)
{
        DDRD |= _BV(DDD7);
        PORTD &= ~_BV(PORTD7);
        DDRD |= _BV(DDD6);
        PORTD &= ~_BV(PORTD6);
        DDRD |= _BV(DDD5);
        PORTD &= ~_BV(PORTD5);
        DDRD |= _BV(DDD4);
        PORTD &= ~_BV(PORTD4);
        DDRD |= _BV(DDD3);
        PORTD &= ~_BV(PORTD3);
        DDRD |= _BV(DDD2);
        PORTD &= ~_BV(PORTD2);
        InicializarUSART();
        ImpedirInterruptUSART();
        InicializarADC();
        Config_Timer1_Interrupt_1Hz();
```

```
sei();
while(1)
{
	if(E1 == 1)
	{
		aux_Temp = 0;
	}
	Dcy = (LerValor8bitADC(2)*0.4) + (Dcy*0.6);
	Dp = (LerValor8bitADC(3)*0.4) + (Dp*0.6);
	B = (LerValor8bitADC(4)*0.4) + (B*0.6);
	S0 = (LerValor8bitADC(5)*0.4) + (S0*0.6);
	S1 = (LerValor8bitADC(6)*0.4) + (S1*0.6);
	if((PFirstCycle == 1))
	{
		E0 = 1;
	}
	if(((E0 == 1) || ((E12 == 1) && (S0 == 255)) || ((E1 == 1))) && (E2 == 0) && (E7 == 0))
	{
		E1 = 1;
	}
	if(((E2 == 1) || ((E1 == 1) && (Dcy == 255) && (B == 0))) && (E3 == 0))
	{
		E2 = 1;
	}
	if(((E3 == 1) || ((E2 == 1) && ((BIgualAa == 1) || (BMaiorQuea == 1)))) && (E4 == 0))
	{
		E3 = 1;
	}
	if(((E4 == 1) || ((E3 == 1) && (B == 0))) && (E5 == 0))
	{
		E4 = 1;
	}
	if(((E5 == 1) || ((E4 == 1) && ((BIgualAb == 1) || (BMaiorQueb == 1)))) && (E6 == 0))
	{
		E5 = 1;
	}
	if(((E6 == 1) || ((E5 == 1) && (B == 0))) && (E10 == 0))
	{
		E6 = 1;
	}
	if(((E7 == 1) || ((E1 == 1) && (Dcy == 255) && (Dp == 0))) && (E8 == 0))
	{
		E7 = 1;
	}
	if(((E8 == 1) || ((E7 == 1) && (Dp == 255))) && (E9 == 0))
	{
		E8 = 1;
	}
	if(((E9 == 1) || ((E8 == 1) && (Dp == 255))) && (E10 == 0))
	{
		E9 = 1;
	}
	if(((E10 == 1) || ((E6 == 1) && (E10 == 1))) && (E11 == 0))
	{
		E10 = 1;
	}
	if(((E11 == 1) || ((E10 == 1) && (Temp == 0) && (S0 == 255))) && (E12 == 0))
	{
		E11 = 1;
	}
	if(((E12 == 1) || ((E11 == 1) && (S1 == 255))) && (E1 == 0))
	{
```

```
		E12 = 1;
	}
	if((E2 == 1))
	{
		if(B > a)
		{
			BMaiorQuea = 1;
			BIgualAa = 0;
			BMenorQuea = 0;
		}
		if(B == a)
		{
			BMaiorQuea = 0;
			BIgualAa = 1;
			BMenorQuea = 0;
		}
		if(B < a)
		{
			BMaiorQuea = 0;
			BIgualAa = 0;
			BMenorQuea = 1;
		}
		PORTD |= _BV(PORTD6);
		PORTD &= ~_BV(PORTD5);
		PORTD &= ~_BV(PORTD4);
		PORTD &= ~_BV(PORTD3);
		PORTD &= ~_BV(PORTD2);
		PORTD &= ~_BV(PORTD1);
	}
	if(((E3 == 1) || ((E5 == 1))))
	{
		PORTD |= _BV(PORTD4);
		PORTD &= ~_BV(PORTD6);
		PORTD &= ~_BV(PORTD5);
		PORTD &= ~_BV(PORTD3);
		PORTD &= ~_BV(PORTD2);
		PORTD &= ~_BV(PORTD1);
	}
	if((E4 == 1))
	{
		if(B > b)
		{
			BMaiorQueb = 1;
			BIgualAb = 0;
			BMenorQueb = 0;
		}
		if(B == b)
		{
			BMaiorQueb = 0;
			BIgualAb = 1;
			BMenorQueb = 0;
		}
		if(B < b)
		{
			BMaiorQueb = 0;
			BIgualAb = 0;
			BMenorQueb = 1;
		}
		PORTD |= _BV(PORTD5);
		PORTD &= ~_BV(PORTD6);
		PORTD &= ~_BV(PORTD4);
		PORTD &= ~_BV(PORTD3);
```

```
		PORTD &= ~_BV(PORTD2);
		PORTD &= ~_BV(PORTD1);
	}
	if(((E7 == 1) || ((E8 == 1))))
	{
		PORTD |= _BV(PORTD3);
		PORTD &= ~_BV(PORTD6);
		PORTD &= ~_BV(PORTD5);
		PORTD &= ~_BV(PORTD4);
		PORTD &= ~_BV(PORTD2);
		PORTD &= ~_BV(PORTD1);
	}
	if((E10 == 1))
	{
		if(aux_Temp == 0)
		{
			Temp = 15;
			aux_Temp = 1;
		}
		PORTD |= _BV(PORTD2);
		PORTD &= ~_BV(PORTD6);
		PORTD &= ~_BV(PORTD5);
		PORTD &= ~_BV(PORTD4);
		PORTD &= ~_BV(PORTD3);
		PORTD &= ~_BV(PORTD1);
	}
	if((E11 == 1))
	{
		PORTD |= _BV(PORTD1);
		PORTD &= ~_BV(PORTD6);
		PORTD &= ~_BV(PORTD5);
		PORTD &= ~_BV(PORTD4);
		PORTD &= ~_BV(PORTD3);
		PORTD &= ~_BV(PORTD2);
	}
	if((E1 == 1))
	{
		E0 = 0;
	}
	if((E2 == 1) && (E7 == 1))
	{
		E1 = 0;
	}
	if((E3 == 1))
	{
		E2 = 0;
	}
	if((E4 == 1))
	{
		E3 = 0;
	}
	if((E5 == 1))
	{
		E4 = 0;
	}
	if((E6 == 1))
	{
		E5 = 0;
	}
	if((E10 == 1))
	{
		E6 = 0;
```

```
                }
                if((E8 == 1))
                {
                        E7 = 0;
                }
                if((E9 == 1))
                {
                        E8 = 0;
                }
                if((E10 == 1))
                {
                        E9 = 0;
                }
                if((E11 == 1))
                {
                        E10 = 0;
                }
                if((E12 == 1))
                {
                        E11 = 0;
                }
                if((E1 == 1))
                {
                        E12 = 0;
                }
        }
}

ISR(TIMER1_COMPA_vect)
{
        if(E10 == 1)
        {
                Temp -= 1;
        }
}

static void Config_Timer1_Interrupt_1Hz(void)
{
        TCCR1A = 0x00;
        TCCR1C = 0x00;
        TCNT1 = 0x00;
        OCR1A = 15624;
        TIFR1 |= _BV(OCF1A);
        TIMSK1 |= _BV(OCIE1A);
        TCCR1B = (_BV(WGM12) | _BV(CS12) | _BV(CS10)) & (~_BV(WGM13) & ~_BV(CS11) & ~_BV(ICNC1) &
~_BV(ICES1));
}
```

Apêndice V – Código em C do código Ladder (método de funções *Set/ Reset*) do 6º Exercício

```
#include "Funcoes_USART.h"
#include "Funcoes_ADC.h"
#include <avr/io.h>
#include <avr/interrupt.h>

static void Config_Timer1_Interrupt_1Hz(void);

uint8_t PFirstCycle = 1;
uint8_t Dcy = 0;
uint8_t Dp = 0;
uint8_t B = 0;
uint8_t S0 = 0;
uint8_t S1 = 0;
uint8_t Temp = 0;
uint8_t aux_Temp = 0;
uint8_t a = 0;
uint8_t b = 0;
uint8_t E0 = 0;
uint8_t E1 = 0;
uint8_t E2 = 0;
uint8_t E3 = 0;
uint8_t E4 = 0;
uint8_t E5 = 0;
uint8_t E6 = 0;
uint8_t E7 = 0;
uint8_t E8 = 0;
uint8_t E9 = 0;
uint8_t E10 = 0;
uint8_t E11 = 0;
uint8_t E12 = 0;
uint8_t BMenorQuea = 0;
uint8_t BIgualAa = 0;
uint8_t BMaiorQuea = 0;
uint8_t BMenorQueb = 0;
uint8_t BIgualAb = 0;
uint8_t BMaiorQueb = 0;

int main (void)
{
        DDRD |= _BV(DDD7);
        PORTD &= ~_BV(PORTD7);
        DDRD |= _BV(DDD6);
        PORTD &= ~_BV(PORTD6);
        DDRD |= _BV(DDD5);
        PORTD &= ~_BV(PORTD5);
        DDRD |= _BV(DDD4);
        PORTD &= ~_BV(PORTD4);
        DDRD |= _BV(DDD3);
        PORTD &= ~_BV(PORTD3);
        DDRD |= _BV(DDD2);
        PORTD &= ~_BV(PORTD2);
        InicializarUSART();
        ImpedirInterruptUSART();
        InicializarADC();
        Config_Timer1_Interrupt_1Hz();
```

```
sei();
while(1)
{
	if(E1 == 1)
	{
		aux_Temp = 0;
	}
	Dcy = (LerValor8bitADC(2)*0.4) + (Dcy*0.6);
	Dp = (LerValor8bitADC(3)*0.4) + (Dp*0.6);
	B = (LerValor8bitADC(4)*0.4) + (B*0.6);
	S0 = (LerValor8bitADC(5)*0.4) + (S0*0.6);
	S1 = (LerValor8bitADC(6)*0.4) + (S1*0.6);
	if((PFirstCycle == 1))
	{
		E0 = 1;
	}
	if((E0 == 1))
	{
		E0 = 0;
		E1 = 1;
	}
	if((E1 == 1) && (Dcy == 255) && (B == 0) && (Dp == 0))
	{
		E1 = 0;
		E7 = 1;
		E2 = 1;
	}
	if(((E2 == 1) && ((BIgualAa == 1) || (BMaiorQuea == 1))))
	{
		E2 = 0;
		E3 = 1;
	}
	if((E3 == 1) && (B == 0))
	{
		E3 = 0;
		E4 = 1;
	}
	if(((E4 == 1) && ((BIgualAb == 1) || (BMaiorQueb == 1))))
	{
		E4 = 0;
		E5 = 1;
	}
	if((E5 == 1) && (B == 0))
	{
		E5 = 0;
		E6 = 1;
	}
	if((E7 == 1) && (Dp == 255))
	{
		E7 = 0;
		E8 = 1;
	}
	if((E8 == 1) && (Dp == 255))
	{
		E8 = 0;
		E9 = 1;
	}
	if((E6 == 1) && (E9 == 1))
	{
		E9 = 0;
		E6 = 0;
		E10 = 1;
```

```
}
if((E10 == 1) && (Temp == 0) && (S0 == 255))
{
		E10 = 0;
		E11 = 1;
}
if((E11 == 1) && (S1 == 255))
{
		E11 = 0;
		E12 = 1;
}
if((E12 == 1) && (S0 == 255))
{
		E12 = 0;
		E1 = 1;
}
if((E2 == 1))
{
		if(B > a)
		{
				BMaiorQuea = 1;
				BIgualAa = 0;
				BMenorQuea = 0;
		}
		if(B == a)
		{
				BMaiorQuea = 0;
				BIgualAa = 1;
				BMenorQuea = 0;
		}
		if(B < a)
		{
				BMaiorQuea = 0;
				BIgualAa = 0;
				BMenorQuea = 1;
		}
		PORTD |= _BV(PORTD6);
		PORTD &= ~_BV(PORTD5);
		PORTD &= ~_BV(PORTD4);
		PORTD &= ~_BV(PORTD3);
		PORTD &= ~_BV(PORTD2);
		PORTD &= ~_BV(PORTD1);
}
if(((E3 == 1) || ((E5 == 1))))
{
		PORTD |= _BV(PORTD4);
		PORTD &= ~_BV(PORTD6);
		PORTD &= ~_BV(PORTD5);
		PORTD &= ~_BV(PORTD3);
		PORTD &= ~_BV(PORTD2);
		PORTD &= ~_BV(PORTD1);
}
if((E4 == 1))
{
		if(B > b)
		{
				BMaiorQueb = 1;
				BIgualAb = 0;
				BMenorQueb = 0;
		}
		if(B == b)
		{
```

```
                                        BMaiorQueb = 0;
                                        BIgualAb = 1;
                                        BMenorQueb = 0;
                                }
                                if(B < b)
                                {
                                        BMaiorQueb = 0;
                                        BIgualAb = 0;
                                        BMenorQueb = 1;
                                }
                                PORTD |= _BV(PORTD5);
                                PORTD &= ~_BV(PORTD6);
                                PORTD &= ~_BV(PORTD4);
                                PORTD &= ~_BV(PORTD3);
                                PORTD &= ~_BV(PORTD2);
                                PORTD &= ~_BV(PORTD1);
                        }
                        if(((E7 == 1) || ((E8 == 1))))
                        {
                                PORTD |= _BV(PORTD3);
                                PORTD &= ~_BV(PORTD6);
                                PORTD &= ~_BV(PORTD5);
                                PORTD &= ~_BV(PORTD4);
                                PORTD &= ~_BV(PORTD2);
                                PORTD &= ~_BV(PORTD1);
                        }
                        if((E10 == 1))
                        {
                                if(aux_Temp == 0)
                                {
                                        Temp = 15;
                                        aux_Temp = 1;
                                }
                                PORTD |= _BV(PORTD2);
                                PORTD &= ~_BV(PORTD6);
                                PORTD &= ~_BV(PORTD5);
                                PORTD &= ~_BV(PORTD4);
                                PORTD &= ~_BV(PORTD3);
                                PORTD &= ~_BV(PORTD1);
                        }
                        if((E11 == 1))
                        {
                                PORTD |= _BV(PORTD1);
                                PORTD &= ~_BV(PORTD6);
                                PORTD &= ~_BV(PORTD5);
                                PORTD &= ~_BV(PORTD4);
                                PORTD &= ~_BV(PORTD3);
                                PORTD &= ~_BV(PORTD2);
                        }
                }
}

ISR(TIMER1_COMPA_vect)
{
        if(E10 == 1)
        {
                Temp -= 1;
        }
}

static void Config_Timer1_Interrupt_1Hz(void)
{
```

```
        TCCR1A = 0x00;
        TCCR1C = 0x00;
        TCNT1 = 0x00;
        OCR1A = 15624;
        TIFR1 |= _BV(OCF1A);
        TIMSK1 |= _BV(OCIE1A);
        TCCR1B = (_BV(WGM12) | _BV(CS12) | _BV(CS10)) & (~_BV(WGM13) & ~_BV(CS11) & ~_BV(ICNC1) &
~_BV(ICES1));
}
```

Anexos

Anexo A – Lista de Exercícios Propostos

EXERCÍCIOS PROPOSTOS

GRAFCET DE NÍVEL 1 e NÍVEL 2

Faça os grafcets de nível 1 e de nível 2 para os seguintes problemas.

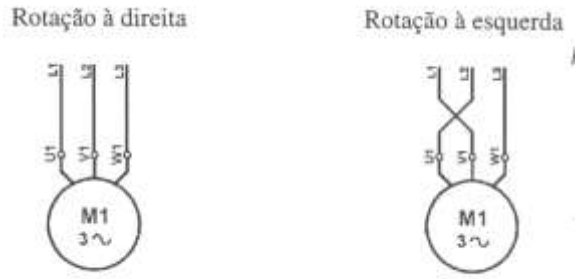
1. INVERSÃO DE MARCHA DE MOTOR TRIFÁSICO

Automatismo que possibilita a inversão de marcha de um motor assíncrono trifásico, em que a marcha à direita acontece quando se pressionar um botão s1 e a marcha à esquerda quando se pressionar um botão s2,

A paragem efectua-se quando se pressionar um botão s0 ou o contacto auxiliar f1 do relé de protecção térmica, que protege o motor contra sobrecargas, fechar.

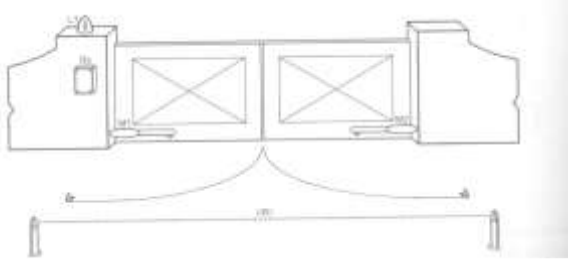
Todos os botões são do tipo pressão e o automatismo possui ainda dois sinalizadores luminosos; um para indicar marcha à direita e outro para indicar marcha à esquerda.

NOTA: A inversão do sentido de rotação de um motor assíncrono trifásico é realizada através da troca de duas das fases que alimentam o motor. Para efectuar esta operação são necessários dois contactores; um que liga o motor com as fases numa determinada sequência e outro que liga com uma sequência, com duas fases trocadas relativamente à primeira sequência. Ver figura seguinte.



2. PORTÃO AUTOMÁTICO

Automatismo para comando à distância de um portão com movimento de abertura/fecho executado por dois motores, MI e M2.



As condições de funcionamento do automatismo são as seguintes:

- A ordem para abertura do portão é fomecida por um emissor (Tx) de comando à distância. Quando o sinal do emissor é recebido pelo receptor (Rx) existente junto ao portão, este abre e uma lâmpada sinalizadora LI acende de forma intermitente;
- No final da abertura do portão são actuados interruptores fim de curso, fel e fc2, colocados, respectivamente, nos êmbolos roscados dos motores MI e M2. Estes fim de curso dão a informação para paragem dos motores MI e M2 e para desligar a lâmpada sinalizadora;
- O portão está aberto durante 60 s, fechando automaticamente no final deste tempo. Durante o fecho, a lâmpada sinalizadora volta a funcionar de forma intermitente;
- A operação de fecho do portão é efectuada durante 20 s, não existindo interruptores fim de curso para detectar o final do fecho;
- Durante o fecho do portão se o emissor do telecomando for novamente pressionado ou a barreira de infravermelhos (IR), existente na parte de dentro do portão, for interrompida, o portão volta a abrir.

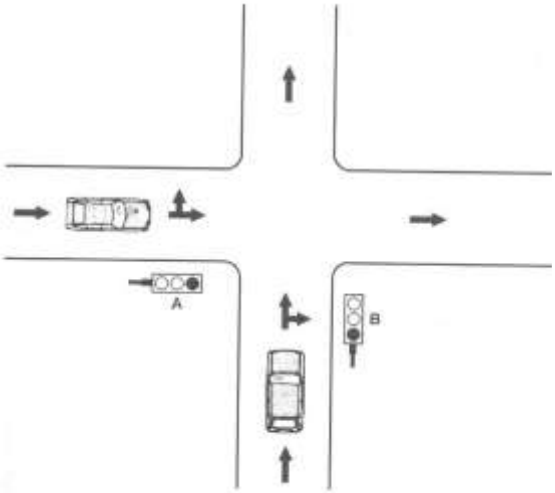
3. SEMÁFOROS

Automatismo para comando de semáforos, designados por A e B, num cruzamento com vias de trânsito num só sentido de acordo com as seguintes condições:

- O sinal vermelho e o verde no semáforo oposto estão ligados durante 30 s;
- Após o sinal verde, acende o sinal amarelo durante 3 s;
- Na passagem do sinal amarelo para vermelho, os sinais vermelhos de ambos os semáforos estão acesos em simultâneo durante 1,5 s;
- Após o sinal vermelho, acende o sinal verde.

Um interruptor il permite ligar/desligar manualmente os semáforos. Quando desligado, o sinal amarelo é colocado em intermitente em ambos os semáforos.

De segunda-feira a quinta-feira, das 00.00h às 06.00h, em que o fluxo de trânsito é reduzido, os semáforos funcionam com o sinal amarelo intermitente, de sexta-feira a domingo, os sinais funcionam durante 24h.



4. TRANSFERÊNCIA DE PEÇAS

Pretende-se transferir peças, entre dois tapetes transportadores, que fazem entre si um ângulo de 90°.

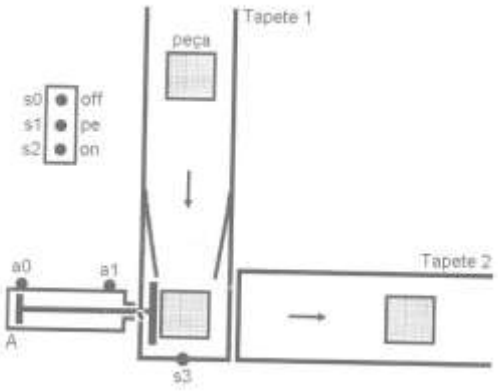
A colocação em funcionamento e paragem dos tapetes é feita através de contactos tipo botão de pressão, respectivamente s2 e s0.

A paragem só deve ocorrer no final de cada ciclo, mas, o sistema também deve ser provido de paragem de emergência (pe) que possibilite, a qualquer momento, por acção sobre um botão (s1) com encravamento, a imediata suspensão das acções em curso.

Pressionado s2, os tapetes entram em movimento. O tapete 1, que transporta peças, coloca-as em frente do detectar s3 que, ao detectá-las, dá ordem de avanço ao cilindro A (A+). Este empurra-as para o tapete 2 até ser actuado o sensor magnético a1 existente no corpo do cilindro.

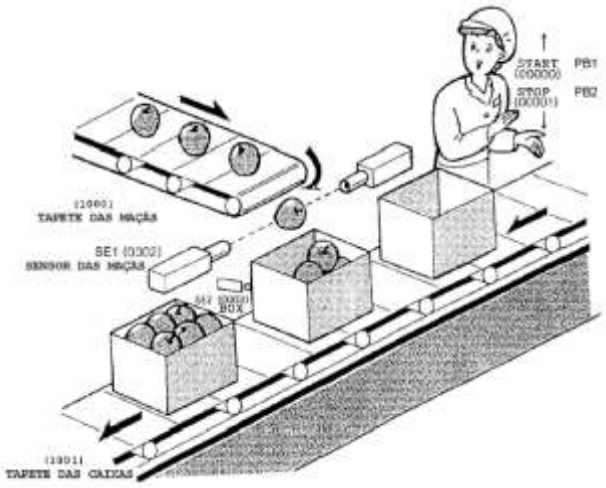
Ao ser actuado o sensor a1, o cilindro A recua (A-) até actuar o sensor a0. Com este sensor actuado, quando chegar uma nova peça, detectada por s3, o ciclo volta a repetir-se.

O cilindro A é de duplo efeito e comandado por electrová1vulas bi-estáveis.



5. ENCAIXOTAMENTO DE MAÇÃS

Pretende-se controlar a linha de encaixotamento de maçãs representada na figura seguinte:

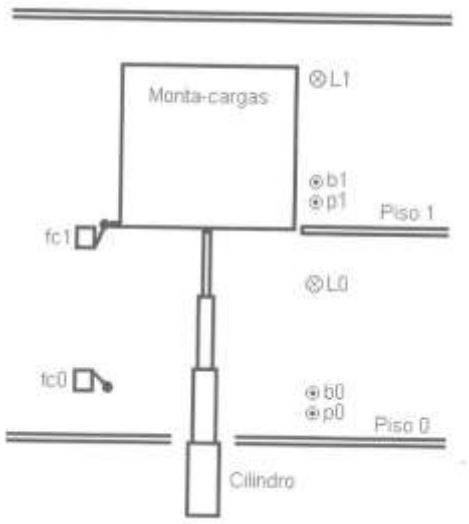


Memória descritiva

Ao sinal de START, o tapete das caixas entra em funcionamento. O sensor das caixas (SE2) ao detetar uma caixa pára este tapete e põe o das maçãs em funcionamento. O sensor das maçãs (SE1) deteta as maçãs que entram na caixa. Após a caixa receber 10 maçãs, o tapete das maçãs pára e o das caixas entra em funcionamento. O sinal de STOP pára todo o processo.

6. MONTA-CARGAS

Um cilindro hidráulico telescópico de duplo efeito, comandado por electroválvulas biestáveis, aciona um monta-cargas que faz o transporte de mercadorias entre dois pisos de uma fábrica. 6



No piso inferior está colocado um botão tipo pressão b0, através do qual se dá a ordem para o monta-cargas subir. No piso superior, um botão b1 do mesmo tipo dá a ordem para o monta-cargas descer. Também existe em cada piso um botão de paragem; p0 no piso 0 e p1 no piso 1, que pressionado provoca a imediata paragem do monta-cargas.

Para além dos botões de pressão, em cada piso, existe ainda uma lâmpada sinalizadora; L0 no piso 0, que pisca quando o monta-cargas está a descer e L1 no piso 1, que pisca quando o monta-cargas está a subir.

A cabina do monta-cargas atua dois interruptores fim de curso; fc0 e fcl, que correspondem, respetivamente, ao final da descida e da subida.

7. GUILHOTINA

Automatismo para comando de uma guilhotina, com lâmina accionada por um cilindro (A) de simples efeito com retomo por mola, cuja operação de corte ocorre nas seguintes condições:

- A lâmina da guilhotina desce e efectua o corte quando o operador carregar e manter pressionados dois botões de pressão (bl e b2), cada um com uma mão, com um tempo de ligação entre ambos não superior a 0,5s;
- Se um dos botões estiver encravado (permanentemente ligado), a lâmina não desce.

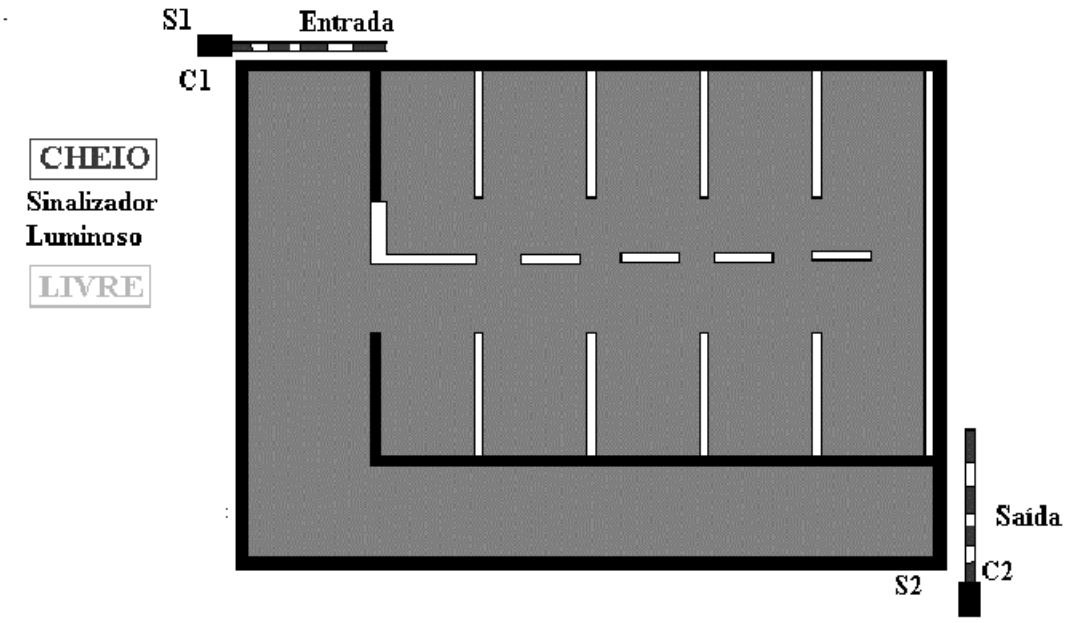
Respeitadas estas condições o cilindro, que comanda a lâmina, empurra esta e é executado o corte (A+). A descida da lâmina termina quando um fim de curso fc for atuado, regressando a lâmina à sua posição de repouso por ação de uma mola.

Nova operação de corte só é possível após ambos os botões deixarem de estar pressionados.

Este funcionamento, que também é comum noutro tipo de máquinas, destina-se a respeitar uma regra de segurança que tem por finalidade reduzir o risco de acidentes, no caso a mutilação das mãos, uma vez que obriga o operador a utilizar as duas mãos para que a guilhotina execute o corte.

8. PARQUE DE ESTACIONAMENTO

Pretende-se implementar um programa no autómato que faça a gestão de um parque de estacionamento:



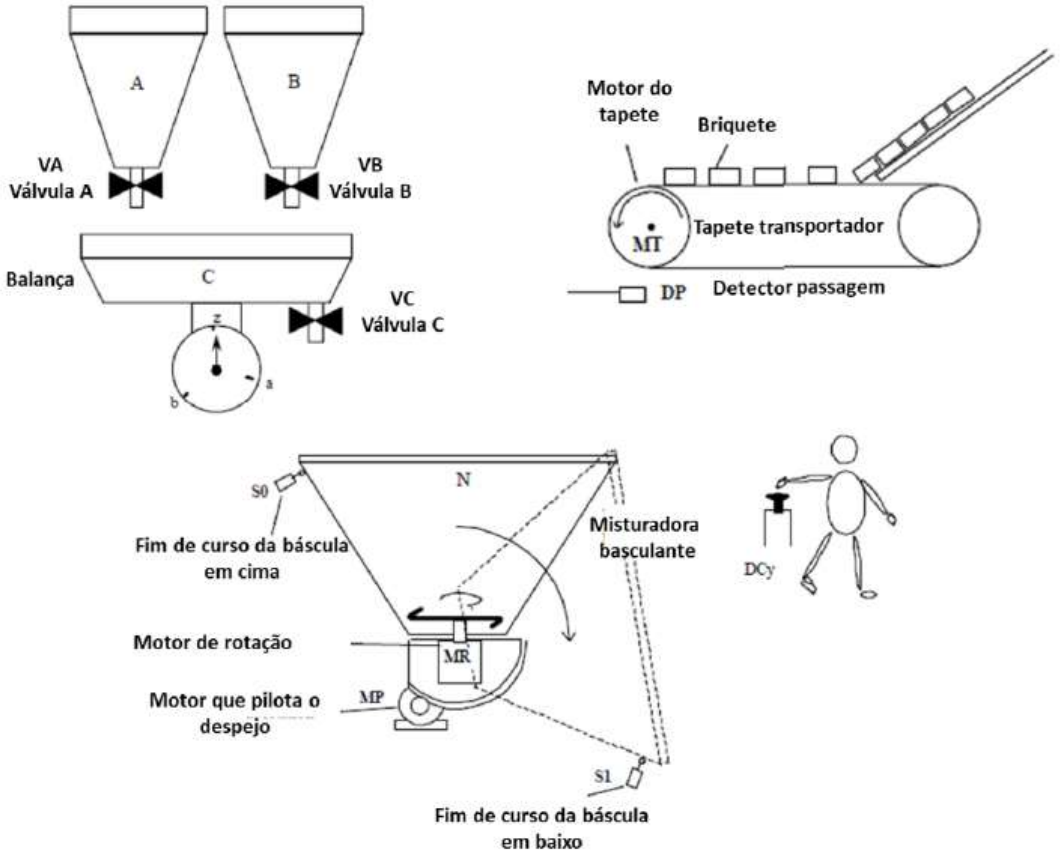
O parque de estacionamento tem capacidade para 10 viaturas. A gestão do número de viaturas no parque será feita com um contador.

O sensor S1 colocado à entrada do parque, ao detetar um automóvel faz atuar C1 cima durante 4 segundos. Quando este tempo terminar e o sensor S1 mudar para OFF, é armado um temporizador com 2 segundos de modo a criar um pequeno atraso. De seguida C1 down é atuado durante 4 segundos. Durante este processo o contador é incrementado. Para a cancela 2 o processo é similar, com a excepção do contador que em vez de ser incrementado é decrementado. Quando o parque está com lotação esgotada o placar luminoso (cheio) deve acender e não pode entrar mais nenhum automóvel, caso contrário existe uma indicação que o parque ainda tem lugares disponíveis.

9. PESAGEM E MISTURA

Considere a seguinte aplicação industrial de **pesagem-mistura** mostrada na figura a seguir:

Dois produtos A e B são previamente pesados na balança C e despejados na misturadora N após a abertura da válvula C. Briquetes soluveis são transportados por um tapete rolante até à misturadora N. Pretende-se misturar os briquetes com uma determinada quantidade dos produtos A e B para se obter uma solução a ser aplicada na indústria téxtil. O processo inicia-se após o operador pressionar o botão de encravamento DCy só podendo ser interrompido após o término do processo. A mistura é composta por 2 briquetes e uma quantidade **a** do produto **A** e uma quantidade **b** do produto **B**. Após esta composição estar feita eles são misturados **durante 15s** após o qual a solução é despejada num recipiente que será posteriormente tratado. O despejo da solução é efetuado pela ação basculante da misturadora N.



10. MÁQUINA DE LAVAR ROUPA

Uma máquina de lavar roupa é controlada por um pequeno PLC. A programação de funcionamento deve obedecer à seguinte especificação:

Fases: Encher, Agitar, Colocar de molho, Lavar, Enxaguar, Centrifugar e Desligar

1. Ao apertar o botão "ligar" a máquina abre uma válvula de enchimento até que um pressostato (sensor de nível alto) atue . Caso isso não aconteça em 12 min, é dado um alarme sonoro de 30 seg e volta ao estado inicial.
2. O motor é ligado em modo agitar. A roupa é agitada durante 5 min. Durante este processo uma bomba de recirculação é posta em funcionamento forçando a água a passar através de um depósito de sabão em pó.
3. A roupa fica em repouso durante 5 min (molho).
4. O motor é novamente ligado em modo de agitação durante 5 min (lavar).
5. O tanque é esvaziado através da abertura de uma válvula de saída (2 min) e do acionamento de uma bomba de sucção durante 1 min para terminar a secagem.
6. O motor é acionado em modo centrifugar durante 10 min com a bomba ligada. A válvula de admissão de água abre-se 3 vezes durante 20 seg na primeira metade deste período.
7. A máquina desliga-se automaticamente.
8. Se o botão desliga for acionado ou a tampa da máquina for aberta a máquina interrompe o seu funcionamento e tem que se começar o processo todo de novo.

EXEMPLOS DE DIAGRAMAS DE ESCADA (LADDER) PARA O AUTÓMATO CPM1 DA OMRON

Instruções básicas (LD, AND,OR, OUT, END, AND LD, OR LD)

1. Pretende-se que o estado da saída 010.00 seja igual ao estado da entrada 000.00.
2. Pretende-se implementar um circuito lógico que active a saída 010.02 do autómato, se as entradas 000.00 e 000.01 e 000.02 estiverem activas (a ON).
3. Pretende-se implementar um circuito lógico que active a saída 010.02 do autómato, se as entradas 000.00 e 000.02 estiverem a OFF e a entrada 000.01estiver a ON.
4. Pretende-se implementar um circuito lógico que active a saída 010.03 do autómato, quando a entrada 000.01 estiver a OFF ou quando as entradas 000.02 ou 000.03 estiverem a ON.
5. Pretende-se implementar um circuito lógico capaz de activar a saída 010.00 sempre que a entrada 000.00 ou 000.01 estiverem a ON e as entradas 000.02 ou 000.03 estiverem também a ON.
6. Pretende-se implementar um circuito lógico capaz de activar a saída 010.00 sempre que as entradas 000.00 e 000.01 ou as entradas 000.02 ou 000.03 estiverem simultaneamente a ON.

Temporizadores e contadores

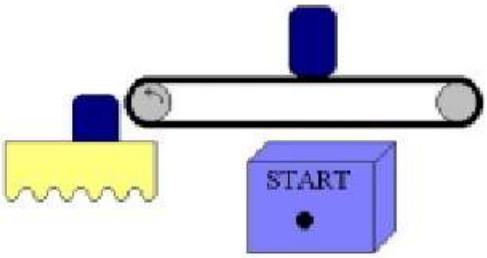
11. Pretende-se implementar um programa que active a saída 010.00, 5 segundos após aactivação da entrada 000.00. Após a activação da saída, a mesma deverá manter-se activa enquanto a entrada estiver a ON.

12. Pretende-se implementar um programa que active a saída 010.00, 7 segundos após a activação da entrada 000.00. Pretende-se implementar este programa recorrendo à técnica de programação de temporizadores em cascata (utilizada quando se pretendeprogramar um temporizador com um tempo superior a 999,9 seg.).

TIM000 = 3 seg.

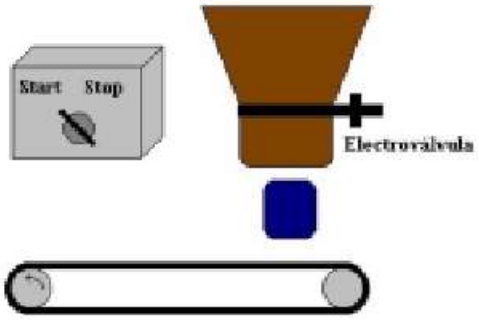
TIM001 = 4 seg.

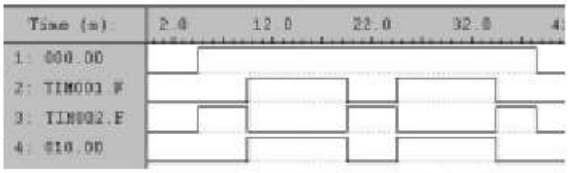
13. Pretende-se implementar um programa que permita ao operador mediante a pressão numa botoneira de START arrancar com um tapete para descarga de um produto. O tapete deve manter-se em movimento durante 5 Seg. por forma a garantir o escoamento do produto.



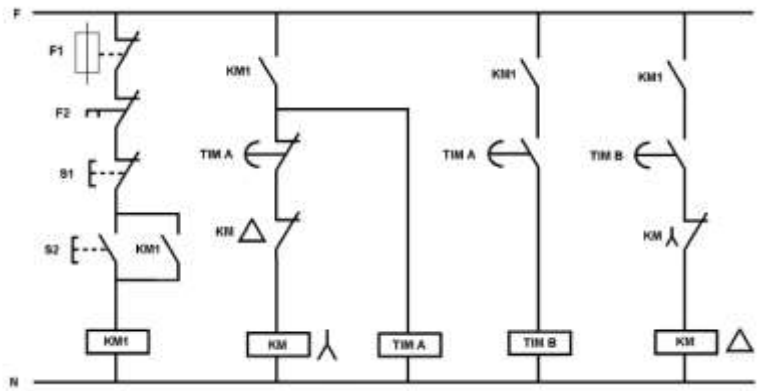
14. Pretende-se implementar um programa que permita ao operador mediante um selector ON/OFF active uma sequência de descarga de produto para um tapete. Para esse efeito a electroválvula existente no silo de descarga deve abrir de 15 em 15 segundos durante 10 segundos.

Diagrama de funcionamento





15. Pretende-se implementar o programa para o arranque em Estrela / Triângulo de um motor de acordo com o esquema eléctrico em anexo.



16. Pretende-se implementar um programa que active a saída 010.03 ao fim de sessenta segundos a partir do momento em que o operador active a entrada 000.01. No caso de existir um corte de energia o programa deve recomeçar a contagem do tempo desde o valor actual na altura do corte de energia.

Instruções de tratamento de dados

1. Pretende-se activar três saídas digitais de acordo com o valor de um contador.

S1 - 010.00 se o valor de contagem for maior que quatro

S2 - 010.01 se o valor de contagem for igual a quatro

S3 - 010.02 se o valor de contagem for menor que quatro

2. Pretende-se transferir o conteúdo de um contador para o canal de saídas físicas do autómato. O contador decrementa de segundo a segundo. Ao atingir o valor zero o contador deverá voltar ao valor de PRESET.

TRADUÇÃO DE GRAFCET PARA LADDER

Traduza os Grafcet efetuados nos 10 primeiros problemas para LADDER.

Utilize a implementação com base nas equações lógicas nos problemas de 1- 5 e as instruções de Set e Reset nos problemas de 6-10.